Temporal_Metrics_PartII_Deep_Learning_Model

December 7, 2023

1 Temporal Metrics Part II Deep Learning Classification Model

Regis University Data Science Practicum II Cammie Newmyer December 7, 2023

2 Introduction

The perception of time is a complex phenomenon, intricately linked to various psychological and physiological conditions. In an endeavor to quantitatively understand and predict altered time perception, the first part of our project involved the mathematical development of a novel constant, termed the Cammie_r constant. This constant emerged from the foundational formula Cammie_r = change in time / (distance/rate), offering a groundbreaking approach to quantifying time perception variations among individuals.

Development of Cammie_r The Cammie_r constant was derived by examining how an average person, given specific conditions, would experience different brainwave types over a 24-hour period. The core idea was to link the time spent in various brainwave states to alterations in time perception. This approach integrated concepts from neurology, specifically the impact of myelination on neural transmission speed. Myelination, a process affecting the insulation of nerve cells, was found to be a crucial element influencing time perception. It directly correlates with speed in the formulas encompassing distance, frequency, cycles, and time.

Application in Machine Learning The initial phase of the project utilized a Random Forest multi-class classification model. This model demonstrated that the increase or decrease in myelination associated with each condition was the most significant feature influencing perceptions of time. Building on these insights, the Cammie_r altered time perception predictor was then applied to a representative 0.001% of the U.S. adult population. In this representative sample, individuals were randomly assigned a corresponding Cammie_r value if they tested positive for a specific condition.

Deep Learning Model Implementation The second part of the project, which this document focuses on, extends the application of the Cammie_r constant using a deep learning model. This model aims to predict an individual's perception of time based on a wide array of conditions and lifestyle factors. The deep learning approach was chosen for its ability to handle complex, non-linear relationships inherent in the dataset. It leverages a comprehensive set of features, including age, sex, mental health conditions, lifestyle factors, and substance use. Each feature is weighted by its associated Cammie_r value, reflecting its influence on time perception.

Results and Insights Our deep learning model, grounded in the insights from the Cammie_r constant, has shown promising results. It accurately predicts the time perception category - "Average," "Slower," or "Faster" - for individuals based on their unique profiles. This model not only stands as a testament to the applicability of the Cammie_r constant in practical scenarios but also opens new avenues in understanding the subjective experience of time.

3 Data Description

Data Acquisition Process The data acquisition process for this model was meticulously designed to ensure a comprehensive and representative dataset, crucial for the effective training and evaluation of the deep learning model. Here's an overview of the steps involved in the data acquisition:

Identifying Relevant Variables: The first step was to identify a range of variables that could influence the perception of time. This included demographic information like age and sex, psychological conditions such as ADHD, Alzheimer's, and anxiety, lifestyle factors like meditation and occupation, as well as substance use (e.g., alcohol, marijuana).

Collection from a Representative Sample: Data was collected from a representative 0.001% of the U.S. adult population. This sample size was chosen to ensure a balance between a dataset that's comprehensive enough for deep learning, while still being manageable in terms of data processing and analysis.

Random Assignment of Cammie_r Values: For each individual in the sample, if they tested positive for a particular condition, they were randomly assigned a corresponding Cammie_r value. This step was crucial in applying the theoretical framework developed in the first part of the project to a practical, predictive model.

Incorporating Myelination Data: Based on the findings from the initial Random Forest model, myelination - the process affecting the insulation of nerve cells - was identified as a key feature influencing time perception. Data regarding the increase or decrease in myelination for each condition were integrated into the dataset.

Standardizing Data for Model Training: The collected data were then standardized for use in the deep learning model. This involved encoding categorical variables, normalizing numerical values, and structuring the data in a format suitable for model input.

Data Features The dataset comprises various features categorized as follows:

Demographic Information: Age, sex. Psychological Conditions: ADHD, Alzheimer's, anxiety, depression, PTSD, etc. Lifestyle Factors: Professional status, meditation practices, sleep patterns. Substance Use: Alcohol, marijuana, prescription medications, and other substances.

Each feature was given a numerical value based on its potential impact on time perception, guided by the Cammie_r constant's framework. The culmination of this data acquisition process was a robust, multi-faceted dataset that serves as the foundation for the deep learning model's predictions on time perception.

4 Theoretical Foundation and Data Estimation

Data Description Theoretical Foundation and Data Estimation In this project, the approach to data acquisition diverges from traditional methods, as it is rooted in a theoretical framework and thought experiment rather than empirical data collection from actual participants. This approach aligns with the innovative nature of the study, which aims to explore the complex interplay between various factors and their influence on the perception of time.

Thought Experiment with ChatGPT 4.0 Utilizing ChatGPT 4.0: The estimation of data was conducted through a thought experiment facilitated by ChatGPT 4.0. This advanced language model, developed by OpenAI, has been trained on a diverse range of texts, enabling it to generate informed estimates about the potential relationships and impacts of various factors on time perception.

Data Estimation Process: Leveraging the vast information contained within ChatGPT 4.0's training data, estimates were made about how different conditions, lifestyle choices, and demographic factors might influence an individual's experience of time. This process involved generating hypothetical data points that reflect the potential outcomes and interactions of these variables.

Theoretical Sample Representation: Rather than collecting data from real individuals, the study conceptualized a representative sample of the U.S. adult population. This theoretical sample was used to assign estimated Cammie_r values to different conditions and factors, based on the model's understanding of their probable impact on myelination and, consequently, on time perception.

Features and Variables: The data encompassed a variety of features, including psychological conditions like ADHD and anxiety, lifestyle factors such as meditation and professional status, and substance use. Each feature was assigned a numerical value, reflecting its estimated impact on time perception as per the Cammie_r constant's framework.

Ethical and Conceptual Considerations Theoretical Nature of the Study: It's crucial to note that this study is theoretical and does not involve real human participants. The data and its implications are hypothetical and are used to explore and model a complex cognitive phenomenon.

Compliance with Ethical Standards: Given the theoretical nature of the study, the typical concerns around data privacy, consent, and ethical clearance for human subjects do not apply. However, the study still aligns with ethical considerations relevant to theoretical and simulation-based research.

Conclusion This unique approach, combining a theoretical exploration with advanced AI capabilities, represents an innovative way to probe into the intricate workings of human perception. While the data and findings are hypothetical, they provide valuable insights and a foundation for future empirical research in this area.

5 Feature Descriptions

The deep learning model in this study uses a range of features, each chosen for its potential impact on the perception of time. These features are divided into several categories, reflecting various aspects of psychological conditions, lifestyle factors, and biological influences. Here's a brief overview of each category:

Demographic Information:

Age: Categorized into groups (e.g., '30_49', '50_69', etc.) to understand how perception of time might vary across different life stages. Sex: Included as a binary variable (Male/Female) to explore any potential differences in time perception based on gender. Psychological Conditions:

Conditions like ADHD, Alzheimer's, Anxiety, Depression, PTSD, Schizophrenia, etc., are included. The model considers how these conditions, which are often associated with alterations in brain function and structure, might influence the subjective experience of time. Lifestyle Factors:

Variables such as Professional Status (e.g., 'Graduate_Student', 'Self_Employed'), Meditation Practices (e.g., 'C_Meditation', 'T_Meditation'), and others. These factors explore the impact of daily activities and mental states on time perception. Sleep patterns, represented by features like 'Insomnia' and 'Ultrasomnia', are also considered, given their known effects on cognitive processes and mental health. Substance Use:

Includes a range of substances like Alcohol, Marijuana, Cocaine, Fentanyl, and Prescription Medications. Substance use can significantly affect neural functioning and, consequently, the perception of time. Cammie_r Values:

Each condition and factor is assigned a Cammie_r value, reflecting its estimated impact on time perception. These values are central to the model's predictions and are derived from the theoretical framework established in the study. Myelination:

Recognized as a key feature in the model, myelination refers to the process of forming a myelin sheath around nerve cells, affecting transmission speed. The model incorporates data on how myelination increases or decreases with each condition, linking it directly to the perceived speed in cognitive processing. Each of these features contributes to the model's ability to predict how an individual perceives time, offering a comprehensive view that integrates biological, psychological, and lifestyle factors. The inclusion of both traditional demographic variables and more nuanced psychological and lifestyle factors underscores the model's holistic approach to understanding time perception.

6 Data Preparation

Import packages for analysis and visualization.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import numpy as np
import pickle

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping

import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

#from google.colab import drive
#drive.mount('/drive/')
```

```
WARNING:tensorflow:From C:\Users\newmy\anaconda3\lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

7 Exploratory Data Analysis

[2]:
```python
# Read the data and verify input
df = pd.read_csv('Temporal_MetricsII_CSV_Main_Dataset_12_2.csv')
df
```

[2]:
```
         AGE SEX  TARGET     MEAN    ADHD  18_29  30_49  50_69  70_89     90+  \
0       18.0   M       1  0.00034 -0.0604      0    0.0    0.0    0.0  0.0000
1       18.0   M       2 -0.00201 -0.0604      0    0.0    0.0    0.0  0.0000
2       18.0   M       2 -0.00039 -0.0604      0    0.0    0.0    0.0  0.0000
3       18.0   M       1  0.00159 -0.0604      0    0.0    0.0    0.0  0.0000
4       18.0   M       1  0.00136 -0.0604      0    0.0    0.0    0.0  0.0000
…        …  ..     …        …       …      …      …      …      …        …
26151  100.0   F       2 -0.00756  0.0000      0    0.0    0.0    0.0 -0.1563
26152  100.0   F       2 -0.00421  0.0000      0    0.0    0.0    0.0 -0.1563
26153  100.0   F       2 -0.00394  0.0000      0    0.0    0.0    0.0 -0.1563
26154  100.0   F       2 -0.00358  0.0000      0    0.0    0.0    0.0 -0.1563
26155  100.0   F       2 -0.00320  0.0000      0    0.0    0.0    0.0 -0.1563

       …  AMATUER_ARTIST  FLOW_STATE  SDT_MINDFULNESS  SDT_NEW_EXPERIENCES  \
0      …             0.0      0.0000           0.0938               0.0000
1      …             0.0      0.0000           0.0000               0.0000
2      …             0.0      0.0313           0.0000               0.0000
3      …             0.0      0.0000           0.0000               0.0708
4      …             0.0      0.0000           0.0000               0.0708
…    …             …          …                …                    …
26151  …             0.0      0.0000           0.0000               0.0000
26152  …             0.0      0.0000           0.0000               0.0000
26153  …             0.0      0.0000           0.0000               0.0000
26154  …             0.0      0.0000           0.0000               0.0000
26155  …             0.0      0.0000           0.0000               0.0000

       SDT_VARY_ACTIVITIES  SDT_REDUCE_STRESS  SDT_LIMIT_MULTITASKING  \
0                   0.0667              0.000                     0.0
1                   0.0667              0.000                     0.0
2                   0.0667              0.000                     0.0
3                   0.0667              0.125                     0.0
4                   0.0667              0.125                     0.0
…                      …                  …                     …
26151               0.0000              0.000                     0.0
26152               0.0000              0.000                     0.0
```

```
26153             0.0000             0.000                     0.0
26154             0.0000             0.000                     0.0
26155             0.0000             0.000                     0.0

       SDT_CREATIVE_PURSUITS  SDT_REFLECT  SDT_SLEEP_NUTRITION
0                        0.0          0.0               0.1563
1                        0.0          0.0               0.1563
2                        0.0          0.0               0.1563
3                        0.0          0.0               0.1563
4                        0.0          0.0               0.1563
…                      …           …                  …
26151                    0.0          0.0               0.0000
26152                    0.0          0.0               0.0000
26153                    0.0          0.0               0.0000
26154                    0.0          0.0               0.0000
26155                    0.0          0.0               0.0000

[26156 rows x 85 columns]
```

[3]:

```python
# Examine the data
df.info()
column_unique_counts = df.nunique()
print(column_unique_counts)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26156 entries, 0 to 26155
Data columns (total 85 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   AGE                            26156 non-null  float64
 1   SEX                            26156 non-null  object
 2   TARGET                         26156 non-null  int64
 3   MEAN                           26156 non-null  float64
 4   ADHD                           26156 non-null  float64
 5   18_29                          26156 non-null  int64
 6   30_49                          26156 non-null  float64
 7   50_69                          26156 non-null  float64
 8   70_89                          26156 non-null  float64
 9   90+                            26156 non-null  float64
 10  ALCOHOL                        26156 non-null  float64
 11  ALZHEIMER'S                    26156 non-null  float64
 12  ANXIETY                        26156 non-null  float64
 13  AUTISM_1                       26156 non-null  float64
 14  AUTISM_2                       26156 non-null  float64
 15  AVERAGE                        26156 non-null  int64
 16  BINGE_WATCHERS                 26156 non-null  float64
 17  BIPOLAR_I                      26156 non-null  float64
 18  BIPOLAR_II                     26156 non-null  float64
```

```
19  CAFFEINE                      26156 non-null  int64
20  CHRONIC_PAIN                  26156 non-null  float64
21  COCAINE                       26156 non-null  float64
22  C_MEDITATION                  26156 non-null  float64
23  DEPRESSION                    26156 non-null  float64
24  DISSOCIATIVE_DISORDER         26156 non-null  float64
25  PROFESSIONAL_ATHLETE          26156 non-null  float64
26  EMERGENCY_EVENT               26156 non-null  int64
27  EPILEPSY                      26156 non-null  float64
28  FENTANYL                      26156 non-null  float64
29  AMATUER_ATHLETE               26156 non-null  float64
30  FULL_TIME_EMPLOYMENT          26156 non-null  int64
31  GAMERS                        26156 non-null  float64
32  GRADUATE_STUDENT              26156 non-null  float64
33  HEROIN                        26156 non-null  float64
34  HIGH_IQ                       26156 non-null  float64
35  LOW_IQ                        26156 non-null  float64
36  KETAMINE                      26156 non-null  float64
37  INSOMNIA                      26156 non-null  float64
38  LSD                           26156 non-null  float64
39  MARIJUANA                     26156 non-null  float64
40  METHAMPHETAMINE               26156 non-null  float64
41  MIGRAINE                      26156 non-null  float64
42  MORPHINE                      26156 non-null  float64
43  MS                            26156 non-null  float64
44  PROFESSIONAL_MUSICIAN         26156 non-null  float64
45  NICOTINE                      26156 non-null  int64
46  ULTRASOMNIA                   26156 non-null  float64
47  ONLINE_SHOPPERS_SURFERS       26156 non-null  float64
48  ONLINE_WORKAHOLICS            26156 non-null  float64
49  OXYCODONE                     26156 non-null  float64
50  PARKINSON'S_DISEASE           26156 non-null  float64
51  PART-TIME_EMPLOYMENT          26156 non-null  int64
52  PERSCRIPTION_ANTIDEPRESSANTS  26156 non-null  float64
53  PERSCRIPTION_SLEEPAIDS        26156 non-null  float64
54  PERSCRIPTION_STIMULANTS       26156 non-null  float64
55  PSILOCYBIN                    26156 non-null  float64
56  PSYCHOSIS                     26156 non-null  float64
57  PTSD                          26156 non-null  float64
58  REMOTE_LEARNERS               26156 non-null  float64
59  RETIREMENT                    26156 non-null  int64
60  SAVANT_1                      26156 non-null  float64
61  SAVANT_2                      26156 non-null  float64
62  SCHIZOPHRENIA                 26156 non-null  float64
63  SELF_EMPLOYED                 26156 non-null  float64
64  SINGLE_WORKING_PARENT         26156 non-null  float64
65  SOCIAL_MEDIA_USERS            26156 non-null  float64
66  STRESS                        26156 non-null  float64
```

```
 67  TBI                            26156 non-null  float64
 68  TERMINAL_1                     26156 non-null  float64
 69  TERMINAL_2                     26156 non-null  float64
 70  T_MEDITATION                   26156 non-null  float64
 71  TR_MEDITATION                  26156 non-null  float64
 72  UNEMPLOYMENT                   26156 non-null  float64
 73  AMATUER_MUSICIAN               26156 non-null  float64
 74  PROFESSIONAL_ARTIST            26156 non-null  float64
 75  AMATUER_ARTIST                 26156 non-null  float64
 76  FLOW_STATE                     26156 non-null  float64
 77  SDT_MINDFULNESS                26156 non-null  float64
 78  SDT_NEW_EXPERIENCES            26156 non-null  float64
 79  SDT_VARY_ACTIVITIES            26156 non-null  float64
 80  SDT_REDUCE_STRESS              26156 non-null  float64
 81  SDT_LIMIT_MULTITASKING         26156 non-null  float64
 82  SDT_CREATIVE_PURSUITS          26156 non-null  float64
 83  SDT_REFLECT                    26156 non-null  float64
 84  SDT_SLEEP_NUTRITION            26156 non-null  float64
dtypes: float64(75), int64(9), object(1)
memory usage: 17.0+ MB
AGE                          83
SEX                           2
TARGET                        3
MEAN                       1231
ADHD                          2
                           …
SDT_REDUCE_STRESS             2
SDT_LIMIT_MULTITASKING        2
SDT_CREATIVE_PURSUITS         2
SDT_REFLECT                   2
SDT_SLEEP_NUTRITION           2
Length: 85, dtype: int64
```

[4]:

```python
any_nans = df.isna().any().any()
print("Are there any NaN values in df?", any_nans)

# Find rows with NaN values
nan_rows = df[df.isna().any(axis=1)]

# Print rows with NaN values
print(nan_rows)
```

```
Are there any NaN values in df? False
Empty DataFrame
Columns: [AGE, SEX, TARGET, MEAN, ADHD, 18_29, 30_49, 50_69, 70_89, 90+,
ALCOHOL, ALZHEIMER'S, ANXIETY, AUTISM_1, AUTISM_2, AVERAGE, BINGE_WATCHERS,
BIPOLAR_I, BIPOLAR_II, CAFFEINE, CHRONIC_PAIN, COCAINE, C_MEDITATION,
DEPRESSION, DISSOCIATIVE_DISORDER, PROFESSIONAL_ATHLETE, EMERGENCY_EVENT,
```

```
EPILEPSY, FENTANYL, AMATUER_ATHLETE, FULL_TIME_EMPLOYMENT, GAMERS,
GRADUATE_STUDENT, HEROIN, HIGH_IQ, LOW_IQ, KETAMINE, INSOMNIA, LSD, MARIJUANA,
METHAMPHETAMINE, MIGRAINE, MORPHINE, MS, PROFESSIONAL_MUSICIAN, NICOTINE,
ULTRASOMNIA, ONLINE_SHOPPERS_SURFERS, ONLINE_WORKAHOLICS, OXYCODONE,
PARKINSON'S_DISEASE, PART-TIME_EMPLOYMENT, PERSCRIPTION_ANTIDEPRESSANTS,
PERSCRIPTION_SLEEPAIDS, PERSCRIPTION_STIMULANTS, PSILOCYBIN, PSYCHOSIS, PTSD,
REMOTE_LEARNERS, RETIREMENT, SAVANT_1, SAVANT_2, SCHIZOPHRENIA, SELF_EMPLOYED,
SINGLE_WORKING_PARENT, SOCIAL_MEDIA_USERS, STRESS, TBI, TERMINAL_1, TERMINAL_2,
T_MEDITATION, TR_MEDITATION, UNEMPLOYMENT, AMATUER_MUSICIAN,
PROFESSIONAL_ARTIST, AMATUER_ARTIST, FLOW_STATE, SDT_MINDFULNESS,
SDT_NEW_EXPERIENCES, SDT_VARY_ACTIVITIES, SDT_REDUCE_STRESS,
SDT_LIMIT_MULTITASKING, SDT_CREATIVE_PURSUITS, SDT_REFLECT, SDT_SLEEP_NUTRITION]
Index: []

[0 rows x 85 columns]
```

8 Exploratory Data Visualization

[5]:
```python
feature_names = df.columns.tolist()
print(feature_names)
```

```
['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '18_29', '30_49', '50_69', '70_89',
'90+', 'ALCOHOL', "ALZHEIMER'S", 'ANXIETY', 'AUTISM_1', 'AUTISM_2', 'AVERAGE',
'BINGE_WATCHERS', 'BIPOLAR_I', 'BIPOLAR_II', 'CAFFEINE', 'CHRONIC_PAIN',
'COCAINE', 'C_MEDITATION', 'DEPRESSION', 'DISSOCIATIVE_DISORDER',
'PROFESSIONAL_ATHLETE', 'EMERGENCY_EVENT', 'EPILEPSY', 'FENTANYL',
'AMATUER_ATHLETE', 'FULL_TIME_EMPLOYMENT', 'GAMERS', 'GRADUATE_STUDENT',
'HEROIN', 'HIGH_IQ', 'LOW_IQ', 'KETAMINE', 'INSOMNIA', 'LSD', 'MARIJUANA',
'METHAMPHETAMINE', 'MIGRAINE', 'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN',
'NICOTINE', 'ULTRASOMNIA', 'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS',
'OXYCODONE', "PARKINSON'S_DISEASE", 'PART-TIME_EMPLOYMENT',
'PERSCRIPTION_ANTIDEPRESSANTS', 'PERSCRIPTION_SLEEPAIDS',
'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN', 'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS',
'RETIREMENT', 'SAVANT_1', 'SAVANT_2', 'SCHIZOPHRENIA', 'SELF_EMPLOYED',
'SINGLE_WORKING_PARENT', 'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1',
'TERMINAL_2', 'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT',
'AMATUER_MUSICIAN', 'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
'SDT_REFLECT', 'SDT_SLEEP_NUTRITION']
```

[6]:
```python
# Assuming df is your DataFrame
all_zero_columns = df.columns[(df == 0).all()].tolist()
print(all_zero_columns)
```

```
['18_29', 'AVERAGE', 'CAFFEINE', 'EMERGENCY_EVENT', 'FULL_TIME_EMPLOYMENT',
'NICOTINE', 'PART-TIME_EMPLOYMENT', 'RETIREMENT']
```

[7]:

```
# List of columns to be dropped
columns_to_drop = ['18_29', 'AVERAGE', 'CAFFEINE', 'EMERGENCY_EVENT',␣
↪'FULL_TIME_EMPLOYMENT', 'NICOTINE', 'PART-TIME_EMPLOYMENT', 'RETIREMENT',␣
↪'AVERAGE.1']

# Dropping the specified columns
subset_df = df.drop(columns=columns_to_drop, errors='ignore')

# Displaying the first few rows of the new DataFrame
print(subset_df.head())
```

```
    AGE SEX  TARGET     MEAN    ADHD  30_49  50_69  70_89  90+  ALCOHOL  …  \
0  18.0   M       1  0.00034 -0.0604    0.0    0.0    0.0  0.0      0.0  …
1  18.0   M       2 -0.00201 -0.0604    0.0    0.0    0.0  0.0      0.0  …
2  18.0   M       2 -0.00039 -0.0604    0.0    0.0    0.0  0.0      0.0  …
3  18.0   M       1  0.00159 -0.0604    0.0    0.0    0.0  0.0      0.0  …
4  18.0   M       1  0.00136 -0.0604    0.0    0.0    0.0  0.0      0.0  …

   AMATUER_ARTIST  FLOW_STATE  SDT_MINDFULNESS  SDT_NEW_EXPERIENCES  \
0             0.0      0.0000           0.0938               0.0000
1             0.0      0.0000           0.0000               0.0000
2             0.0      0.0313           0.0000               0.0000
3             0.0      0.0000           0.0000               0.0708
4             0.0      0.0000           0.0000               0.0708

   SDT_VARY_ACTIVITIES  SDT_REDUCE_STRESS  SDT_LIMIT_MULTITASKING  \
0               0.0667              0.000                     0.0
1               0.0667              0.000                     0.0
2               0.0667              0.000                     0.0
3               0.0667              0.125                     0.0
4               0.0667              0.125                     0.0

   SDT_CREATIVE_PURSUITS  SDT_REFLECT  SDT_SLEEP_NUTRITION
0                    0.0          0.0               0.1563
1                    0.0          0.0               0.1563
2                    0.0          0.0               0.1563
3                    0.0          0.0               0.1563
4                    0.0          0.0               0.1563

[5 rows x 77 columns]
```

[8]:

```
subset_feature_names = subset_df.columns.tolist()
print(subset_feature_names)
```

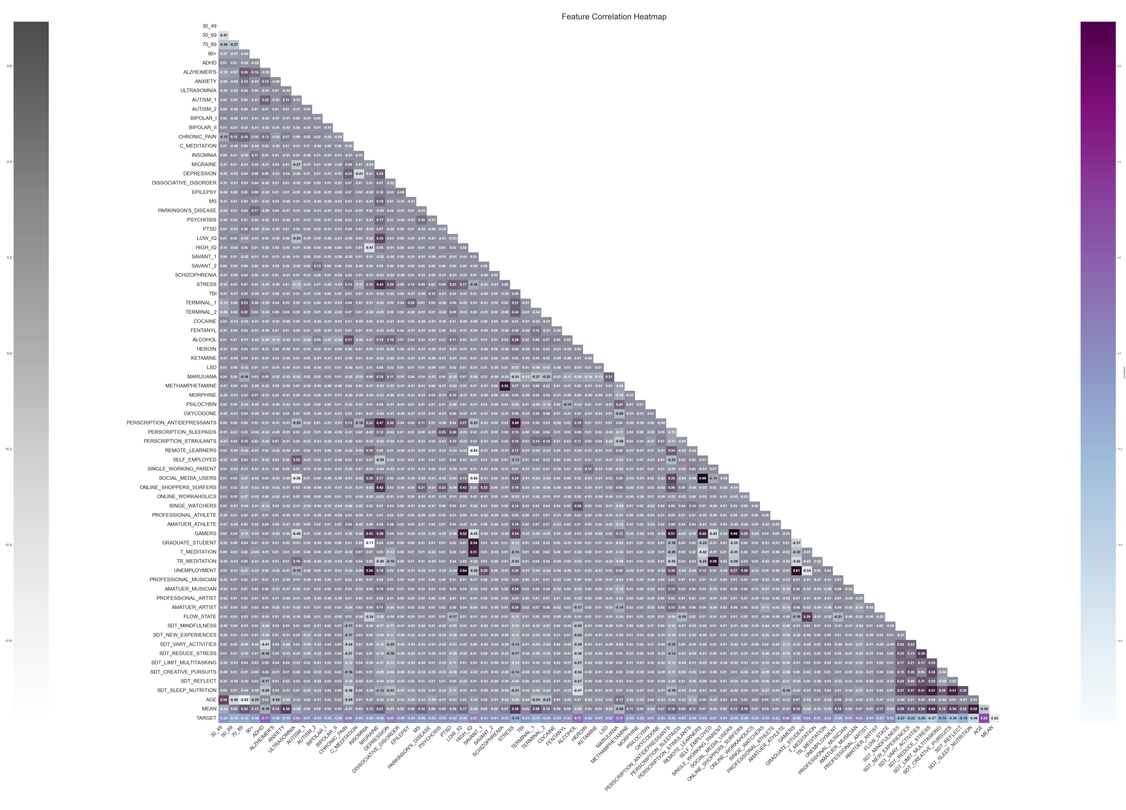
```
['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '30_49', '50_69', '70_89', '90+',
'ALCOHOL', "ALZHEIMER'S", 'ANXIETY', 'AUTISM_1', 'AUTISM_2', 'BINGE_WATCHERS',
'BIPOLAR_I', 'BIPOLAR_II', 'CHRONIC_PAIN', 'COCAINE', 'C_MEDITATION',
'DEPRESSION', 'DISSOCIATIVE_DISORDER', 'PROFESSIONAL_ATHLETE', 'EPILEPSY',
```

```
'FENTANYL', 'AMATUER_ATHLETE', 'GAMERS', 'GRADUATE_STUDENT', 'HEROIN',
'HIGH_IQ', 'LOW_IQ', 'KETAMINE', 'INSOMNIA', 'LSD', 'MARIJUANA',
'METHAMPHETAMINE', 'MIGRAINE', 'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN',
'ULTRASOMNIA', 'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS', 'OXYCODONE',
"PARKINSON'S_DISEASE", 'PERSCRIPTION_ANTIDEPRESSANTS', 'PERSCRIPTION_SLEEPAIDS',
'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN', 'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS',
'SAVANT_1', 'SAVANT_2', 'SCHIZOPHRENIA', 'SELF_EMPLOYED',
'SINGLE_WORKING_PARENT', 'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1',
'TERMINAL_2', 'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT',
'AMATUER_MUSICIAN', 'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
'SDT_REFLECT', 'SDT_SLEEP_NUTRITION']
```

[9]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Specify the order of columns manually
ordered_columns = ['30_49', '50_69', '70_89', '90+', 'ADHD', "ALZHEIMER'S",␣
 ↪'ANXIETY', 'ULTRASOMNIA',
                   'AUTISM_1', 'AUTISM_2', 'BIPOLAR_I', 'BIPOLAR_II',␣
 ↪'CHRONIC_PAIN', 'C_MEDITATION', 'INSOMNIA', 'MIGRAINE',
                   'DEPRESSION', 'DISSOCIATIVE_DISORDER', 'EPILEPSY', 'MS',␣
 ↪"PARKINSON'S_DISEASE", 'PSYCHOSIS', 'PTSD', 'LOW_IQ', 'HIGH_IQ',
                   'SAVANT_1', 'SAVANT_2', 'SCHIZOPHRENIA', 'STRESS', 'TBI',␣
 ↪'TERMINAL_1', 'TERMINAL_2',
                   'COCAINE',  'FENTANYL', 'ALCOHOL', 'HEROIN', 'KETAMINE',␣
 ↪'LSD', 'MARIJUANA', 'METHAMPHETAMINE', 'MORPHINE',  'PSILOCYBIN',
                   'OXYCODONE', 'PERSCRIPTION_ANTIDEPRESSANTS',␣
 ↪'PERSCRIPTION_SLEEPAIDS', 'PERSCRIPTION_STIMULANTS', 'REMOTE_LEARNERS',
                   'SELF_EMPLOYED', 'SINGLE_WORKING_PARENT',␣
 ↪'SOCIAL_MEDIA_USERS', 'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS',
                   'BINGE_WATCHERS', 'PROFESSIONAL_ATHLETE', 'AMATUER_ATHLETE',␣
 ↪'GAMERS', 'GRADUATE_STUDENT',
                   'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT',␣
 ↪'PROFESSIONAL_MUSICIAN', 'AMATUER_MUSICIAN',
                   'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',␣
 ↪'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
                   'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING',␣
 ↪'SDT_CREATIVE_PURSUITS', 'SDT_REFLECT', 'SDT_SLEEP_NUTRITION',
                   'AGE', 'SEX','MEAN','TARGET']

# Reorder DataFrame columns
df_ordered = subset_df[ordered_columns]

# Create the correlation matrix
```

```python
corr_matrix = df_ordered.corr()

corr_matrix.to_excel('ordered_correlation_matrix.xlsx')

# Create a mask to show the bottom row and hide the upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Set the style to remove the gray background
sns.set_style("white")

# Create a figure and axes for the heatmaps
fig, ax = plt.subplots(figsize=(100, 50))

# Plot the original heatmap with the modified mask
heatmap = sns.heatmap(corr_matrix, mask=mask, xticklabels=[col if col != 'TARGET' else '' for col in corr_matrix.columns],
                      yticklabels=[col for col in corr_matrix.columns],
                      annot=True, fmt='.2f', cmap='BuPu', linewidths=0, linecolor=None,
                      cbar_kws={'location': 'right', 'label': 'Correlation'}, ax=ax)

# Rotate the x-axis tick labels by 45 degrees
plt.setp(ax.get_xticklabels(), rotation=45, ha='right')

# Create a new mask to show only the bottom row and completely hide the upper triangle
bottom_row_mask = np.zeros_like(corr_matrix, dtype=bool)
bottom_row_mask[-1, :-1] = True

# Plot the new heatmap with only the bottom row and completely hidden upper triangle
sns.heatmap(corr_matrix, mask=np.logical_or(mask, bottom_row_mask),
            xticklabels=[col if col != 'TARGET' else '' for col in corr_matrix.columns],
            yticklabels=[col for col in corr_matrix.columns],
            annot=True, fmt='.2f', cmap='Greys', linewidths=0, linecolor=None,
            cbar_kws={'location': 'left', 'pad': 0.15}, ax=ax, alpha=0.7)
plt.draw()

# Now, explicitly set the rotation of x-axis and y-axis tick labels
plt.setp(ax.get_xticklabels(), rotation=45, ha='right')
plt.setp(ax.get_yticklabels(), rotation=0)

# Remove upper triangle annotations
for i in range(len(corr_matrix.columns)):
```

```python
    for j in range(i+1, len(corr_matrix.columns)):
        ax.text(j + 0.5, i + 0.5, '', ha='center', va='center')

# Modify the font size for annotations
for text in ax.texts:
    text.set_fontsize(14)
    text.set_weight('bold')

# Modify the font size for tick labels
ax.tick_params(axis='both', which='both', labelsize=20)

# Modify the colorbar's label font size
cbar = ax.collections[1].colorbar
cbar.ax.tick_params(labelsize=14)

# Add a title with padding and larger font
ax.set_title('Feature Correlation Heatmap', fontsize=32, pad=10)

plt.show()
```



[10]:
```python
# Get the list of selected features (column names)
selected_feature_list = subset_df.columns.tolist()
```

```
# Find pairs of features with the desired correlation range
selected_features = set()
for i in corr_matrix.columns:
    for j in corr_matrix.index:
        if 0.5 <= abs(corr_matrix.loc[i, j]) <= 1 and i != j:
            selected_features.update([i, j])

# Create a new DataFrame with selected features
subset_CM_df = subset_df[list(selected_features)]
# Print the list of selected features
print("Selected Features:", selected_feature_list)
```

```
Selected Features: ['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '30_49', '50_69',
'70_89', '90+', 'ALCOHOL', "ALZHEIMER'S", 'ANXIETY', 'AUTISM_1', 'AUTISM_2',
'BINGE_WATCHERS', 'BIPOLAR_I', 'BIPOLAR_II', 'CHRONIC_PAIN', 'COCAINE',
'C_MEDITATION', 'DEPRESSION', 'DISSOCIATIVE_DISORDER', 'PROFESSIONAL_ATHLETE',
'EPILEPSY', 'FENTANYL', 'AMATUER_ATHLETE', 'GAMERS', 'GRADUATE_STUDENT',
'HEROIN', 'HIGH_IQ', 'LOW_IQ', 'KETAMINE', 'INSOMNIA', 'LSD', 'MARIJUANA',
'METHAMPHETAMINE', 'MIGRAINE', 'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN',
'ULTRASOMNIA', 'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS', 'OXYCODONE',
"PARKINSON'S_DISEASE", 'PERSCRIPTION_ANTIDEPRESSANTS', 'PERSCRIPTION_SLEEPAIDS',
'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN', 'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS',
'SAVANT_1', 'SAVANT_2', 'SCHIZOPHRENIA', 'SELF_EMPLOYED',
'SINGLE_WORKING_PARENT', 'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1',
'TERMINAL_2', 'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT',
'AMATUER_MUSICIAN', 'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
'SDT_REFLECT', 'SDT_SLEEP_NUTRITION']
```

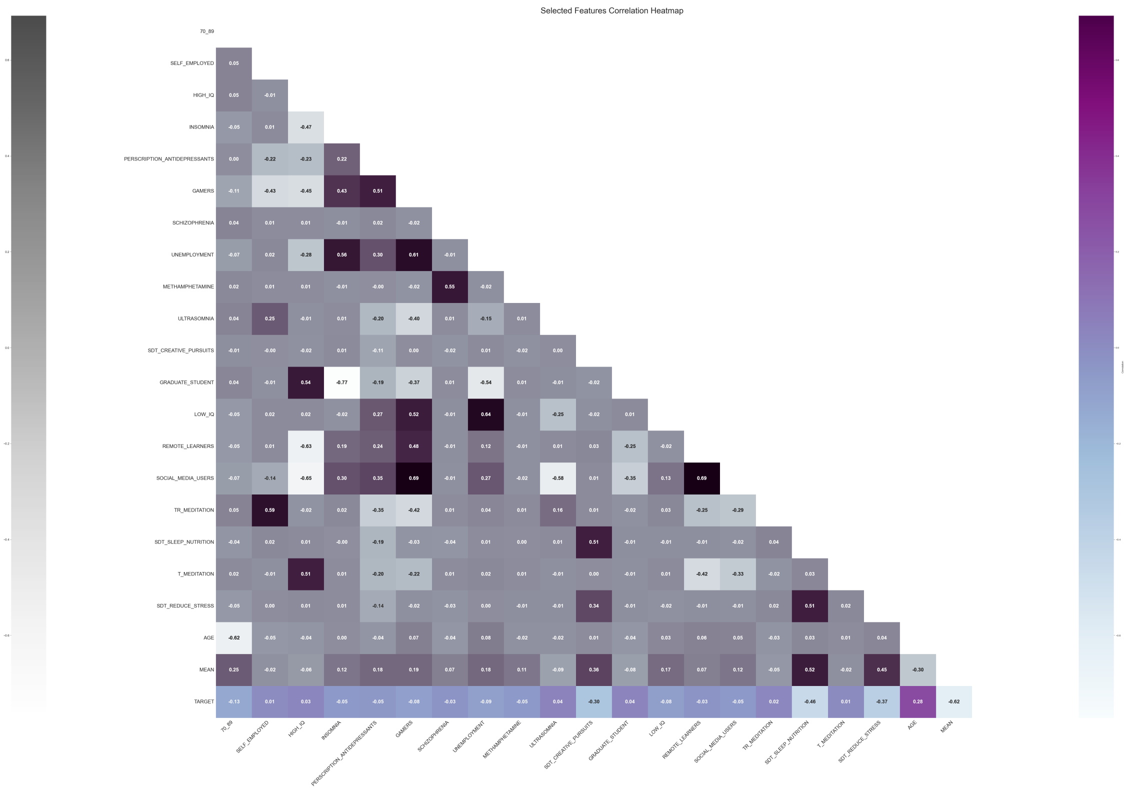
[11]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Specify the order of columns manually
selected_features_columns = ['70_89', 'SELF_EMPLOYED', 'HIGH_IQ', 'INSOMNIA',␣
 ↪'PERSCRIPTION_ANTIDEPRESSANTS',
                             'GAMERS', 'SCHIZOPHRENIA', 'UNEMPLOYMENT',␣
 ↪'METHAMPHETAMINE', 'ULTRASOMNIA',
                             'SDT_CREATIVE_PURSUITS', 'GRADUATE_STUDENT',␣
 ↪'LOW_IQ', 'REMOTE_LEARNERS',
                             'SOCIAL_MEDIA_USERS', 'TR_MEDITATION',␣
 ↪'SDT_SLEEP_NUTRITION', 'T_MEDITATION',
                             'SDT_REDUCE_STRESS', 'AGE', 'MEAN', 'TARGET']
```

```
## Reorder DataFrame columns
df_selected = subset_CM_df[selected_features_columns]

# Check the order of columns
print("Ordered Columns in df_selected:", df_selected.columns.tolist())

# Create the correlation matrix
corr_matrix_selected = df_selected.corr()

# Check the order of columns in the correlation matrix
print("Ordered Columns in corr_matrix_selected:", corr_matrix_selected.columns.
 ↪tolist())

# Create the correlation matrix
corr_matrix_selected = df_selected.corr()

corr_matrix_selected.to_excel('selected_features_correlation_matrix.xlsx')

# Create a mask to show the bottom row and hide the upper triangle
mask = np.triu(np.ones_like(corr_matrix_selected, dtype=bool))

# Set the style to remove the gray background
sns.set_style("white")

# Create a figure and axes for the heatmaps
fig, ax = plt.subplots(figsize=(100, 50))

# Plot the original heatmap with the modified mask
heatmap = sns.heatmap(corr_matrix_selected, mask=mask, xticklabels=[col if col !
 ↪= 'TARGET' else '' for col in corr_matrix_selected.columns],
                      yticklabels=[col for col in corr_matrix_selected.columns],
                      annot=True, fmt='.2f', cmap='BuPu', linewidths=0,␣
 ↪linecolor=None,
                      cbar_kws={'location': 'right', 'label': 'Correlation'},␣
 ↪ax=ax)

# Rotate the x-axis tick labels by 45 degrees
plt.setp(ax.get_xticklabels(), rotation=45, ha='right')

# Create a new mask to show only the bottom row and completely hide the upper␣
 ↪triangle
bottom_row_mask = np.zeros_like(corr_matrix_selected, dtype=bool)
bottom_row_mask[-1, :-1] = True

# Plot the new heatmap with only the bottom row and completely hidden upper␣
 ↪triangle
```

```
sns.heatmap(corr_matrix_selected, mask=np.logical_or(mask, bottom_row_mask),
            xticklabels=[col if col != 'TARGET' else '' for col in␣
 ↪corr_matrix_selected.columns],
            yticklabels=[col for col in corr_matrix_selected.columns],
            annot=True, fmt='.2f', cmap='Greys', linewidths=0, linecolor=None,
            cbar_kws={'location': 'left', 'pad': 0.15}, ax=ax, alpha=0.7)
plt.draw()

# Now, explicitly set the rotation of x-axis and y-axis tick labels
plt.setp(ax.get_xticklabels(), rotation=45, ha='right')
plt.setp(ax.get_yticklabels(), rotation=0)

# Remove upper triangle annotations
for i in range(len(corr_matrix_selected.columns)):
    for j in range(i+1, len(corr_matrix_selected.columns)):
        ax.text(j + 0.5, i + 0.5, '', ha='center', va='center')

# Modify the font size for annotations
for text in ax.texts:
    text.set_fontsize(20)
    text.set_weight('bold')

# Modify the font size for tick labels
ax.tick_params(axis='both', which='both', labelsize=20)

# Modify the colorbar's label font size
cbar = ax.collections[1].colorbar
cbar.ax.tick_params(labelsize=12)

# Add a title with padding and larger font
ax.set_title('Selected Features Correlation Heatmap', fontsize=32, pad=10)

plt.show()
```

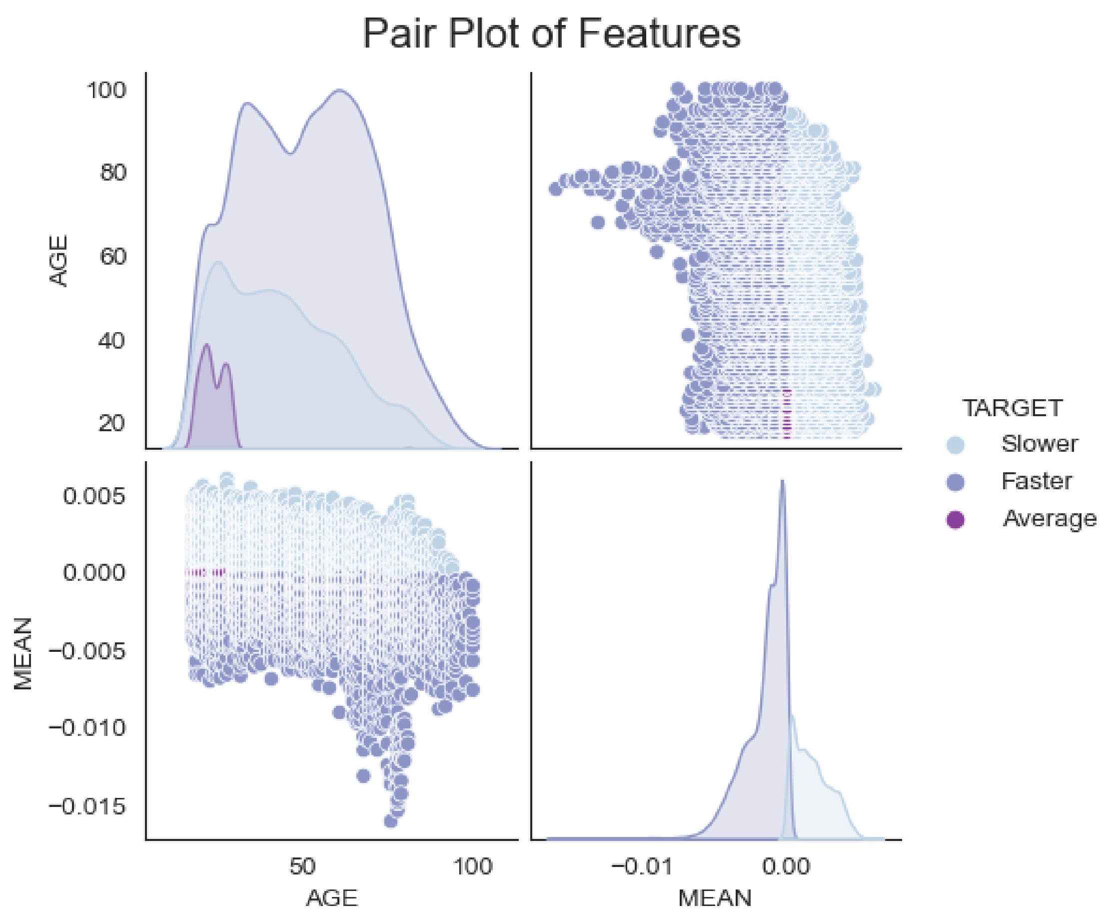
```
Ordered Columns in df_selected: ['70_89', 'SELF_EMPLOYED', 'HIGH_IQ',
'INSOMNIA', 'PERSCRIPTION_ANTIDEPRESSANTS', 'GAMERS', 'SCHIZOPHRENIA',
'UNEMPLOYMENT', 'METHAMPHETAMINE', 'ULTRASOMNIA', 'SDT_CREATIVE_PURSUITS',
'GRADUATE_STUDENT', 'LOW_IQ', 'REMOTE_LEARNERS', 'SOCIAL_MEDIA_USERS',
'TR_MEDITATION', 'SDT_SLEEP_NUTRITION', 'T_MEDITATION', 'SDT_REDUCE_STRESS',
'AGE', 'MEAN', 'TARGET']
Ordered Columns in corr_matrix_selected: ['70_89', 'SELF_EMPLOYED', 'HIGH_IQ',
'INSOMNIA', 'PERSCRIPTION_ANTIDEPRESSANTS', 'GAMERS', 'SCHIZOPHRENIA',
'UNEMPLOYMENT', 'METHAMPHETAMINE', 'ULTRASOMNIA', 'SDT_CREATIVE_PURSUITS',
'GRADUATE_STUDENT', 'LOW_IQ', 'REMOTE_LEARNERS', 'SOCIAL_MEDIA_USERS',
'TR_MEDITATION', 'SDT_SLEEP_NUTRITION', 'T_MEDITATION', 'SDT_REDUCE_STRESS',
'AGE', 'MEAN', 'TARGET']
```



9 VISUALIZATIONS AND EDA

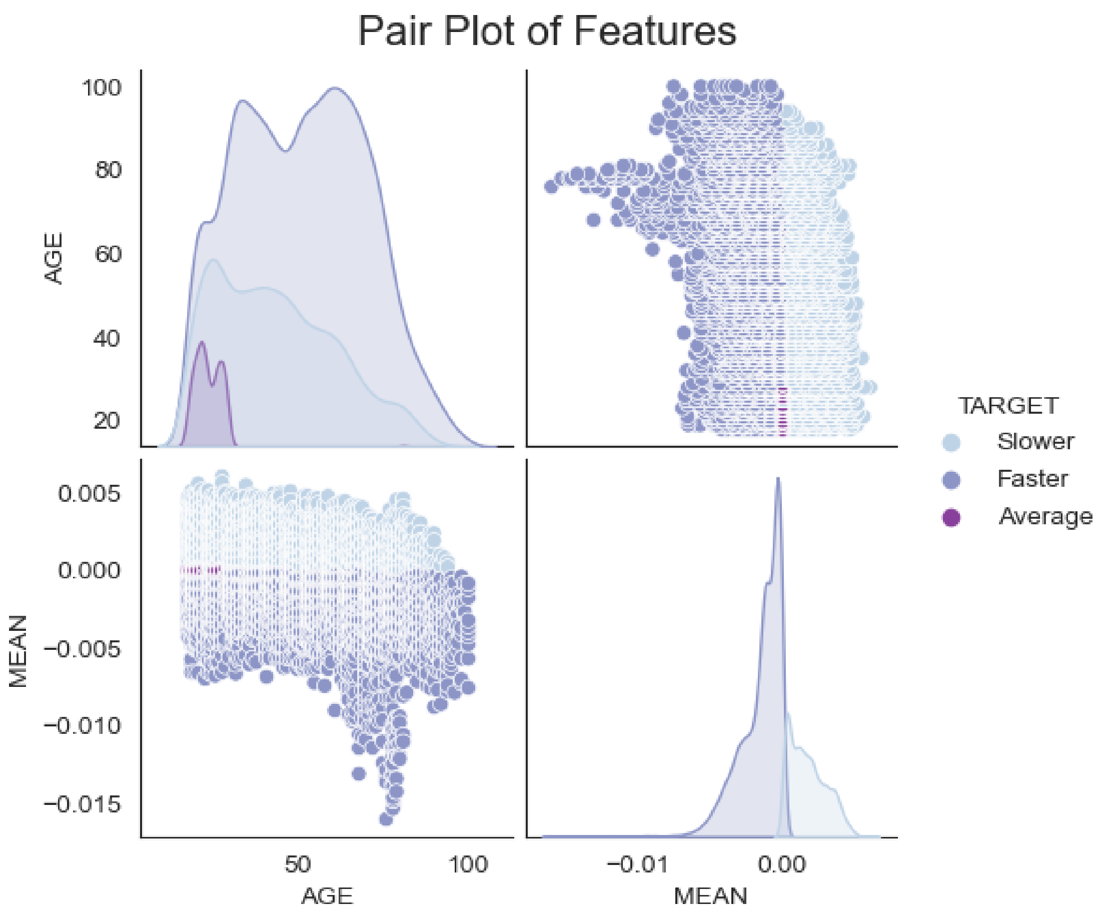
[12]:
```
# Select the columns for the pair plot
pairplot_columns = ['AGE', 'SEX', 'TARGET', 'MEAN']
# Subset the data with the selected columns
pairplot_data = subset_df[pairplot_columns]

# Set the hue to 'Target' for color differentiation
pairplot_data['TARGET'] = pairplot_data['TARGET'].map({0: 'Average', 1:␣
↪'Slower', 2: 'Faster'})

# Create the pair plot with labeled axes and blue hue
pairplot = sns.pairplot(pairplot_data, hue='TARGET', palette='BuPu')

# Add a title to the pair plot
pairplot.fig.suptitle('Pair Plot of Features', fontsize=16)

# Adjust the spacing between subplots to avoid overlap
pairplot.tight_layout()

# Display the plot
```

```
plt.show()
```



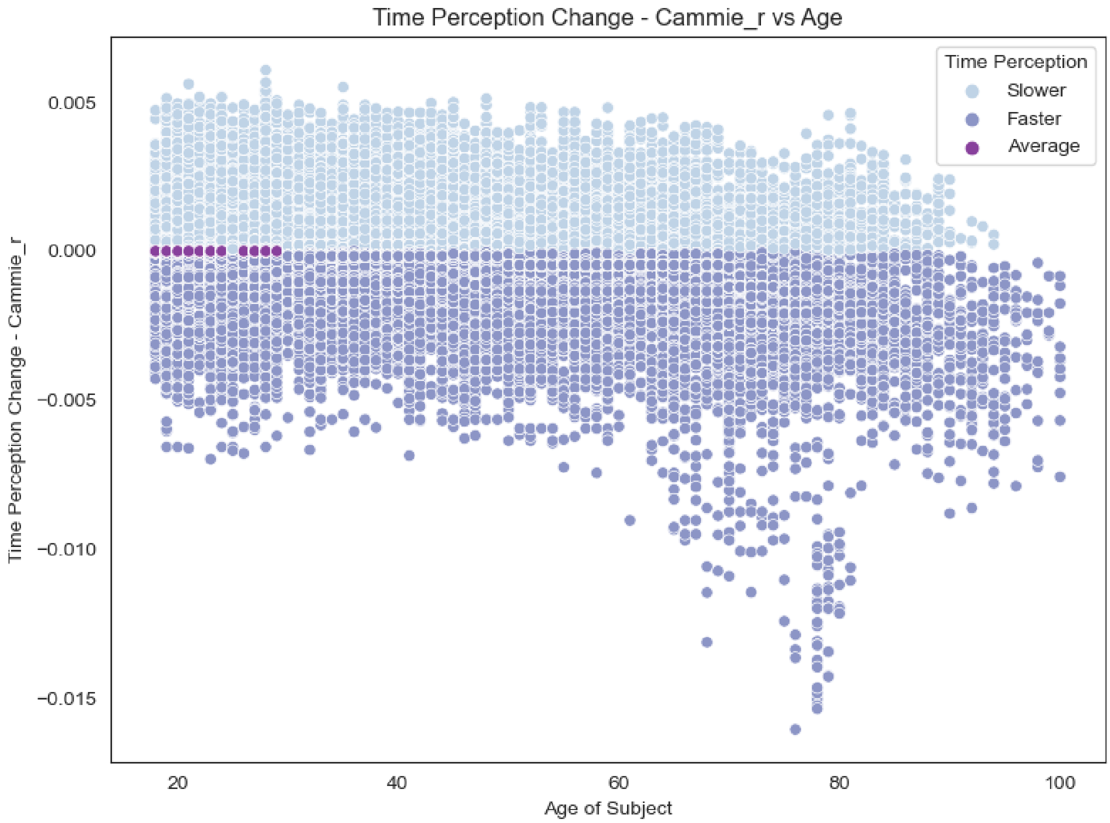

```
[13]: # Select the columns for the pair plot
pairplot_columns = ['AGE', 'TARGET', 'MEAN']
# Subset the data with the selected columns
pairplot_data = df_selected[pairplot_columns]

# Set the hue to 'Target' for color differentiation
pairplot_data['TARGET'] = pairplot_data['TARGET'].map({0: 'Average', 1:
 ↪'Slower', 2: 'Faster'})

# Create the pair plot with labeled axes and blue hue
pairplot = sns.pairplot(pairplot_data, hue='TARGET', palette='BuPu')

# Add a title to the pair plot
pairplot.fig.suptitle('Pair Plot of Features', fontsize=16)

# Adjust the spacing between subplots to avoid overlap
```

```
pairplot.tight_layout()

# Display the plot
plt.show()
```



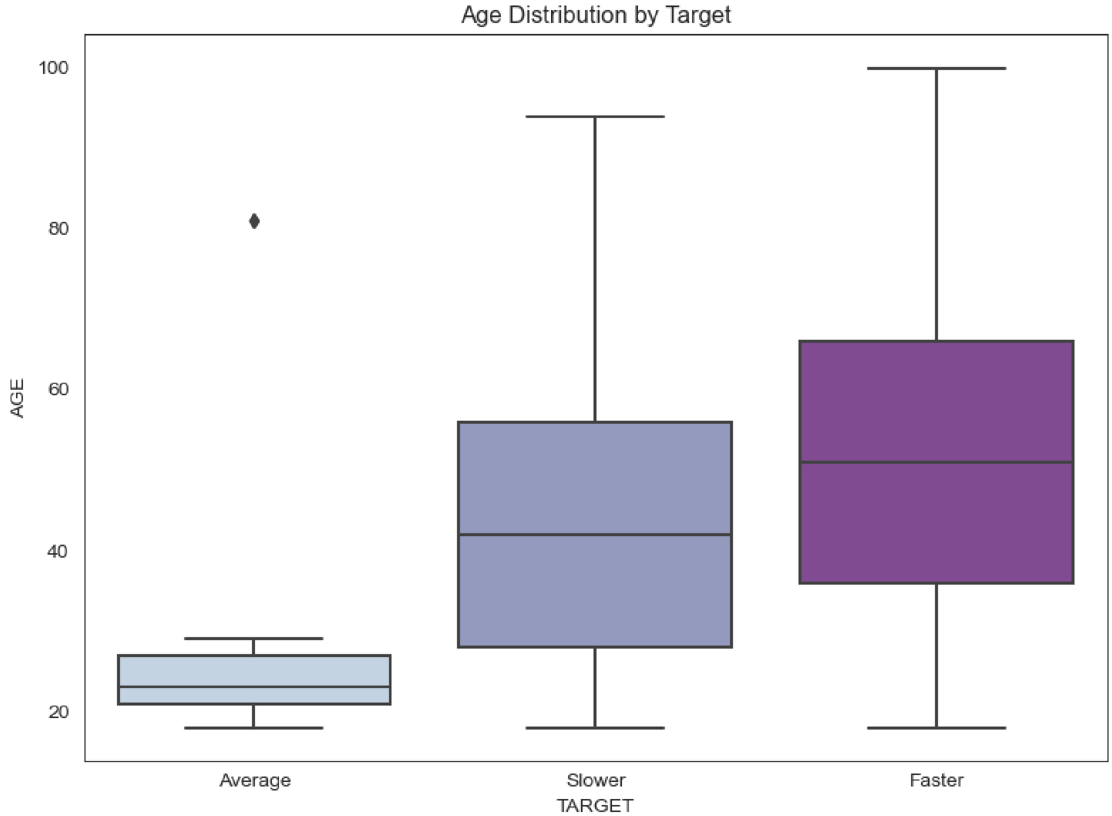

```
[14]: # Use target names
target_names = {0: "Average", 1: "Slower", 2: "Faster"}
plot_df = subset_CM_df.copy()
plot_df['TARGET']=plot_df['TARGET'].replace(target_names)

# Scatter Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='AGE', y='MEAN', hue='TARGET', data=plot_df, palette='BuPu')
plt.xlabel('Age of Subject')
plt.ylabel('Time Perception Change - Cammie_r')
plt.title('Time Perception Change - Cammie_r vs Age')
plt.tight_layout()
plt.legend(title='Time Perception')
```

```
plt.show()
```



[15]:
```
# Box Plot
target_names = {
    0: 'Average',
    1: 'Slower',
    2: 'Faster',
    # Add more mappings as required
}

# Now, use this mapping to replace the numerical values in your plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='TARGET', y='AGE', data=subset_df, palette='BuPu')

# Replace x-tick labels
plt.xticks(ticks=range(len(target_names)), labels=target_names.values())

plt.xlabel('TARGET')
plt.ylabel('AGE')
plt.title('Age Distribution by Target')
```

```
plt.tight_layout()
plt.show()
```



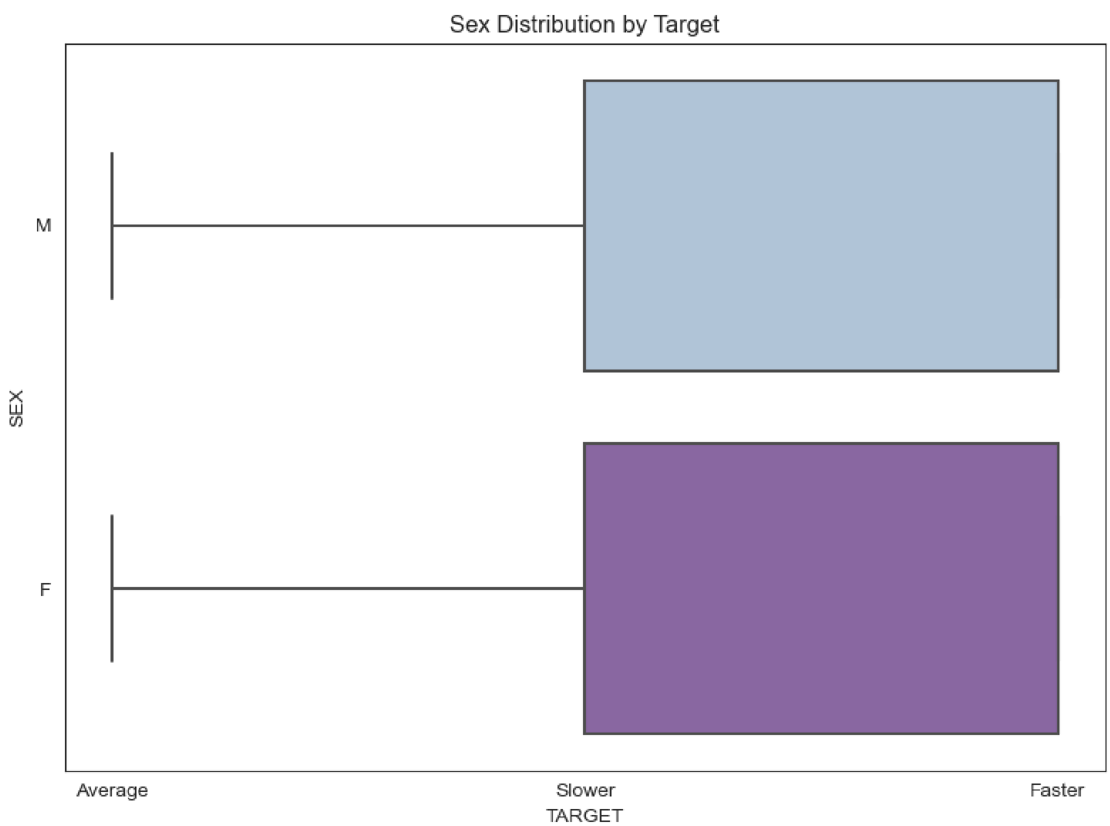
[16]:
```
print(subset_df.columns)
```

```
Index(['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '30_49', '50_69', '70_89',
       '90+', 'ALCOHOL', 'ALZHEIMER'S', 'ANXIETY', 'AUTISM_1', 'AUTISM_2',
       'BINGE_WATCHERS', 'BIPOLAR_I', 'BIPOLAR_II', 'CHRONIC_PAIN', 'COCAINE',
       'C_MEDITATION', 'DEPRESSION', 'DISSOCIATIVE_DISORDER',
       'PROFESSIONAL_ATHLETE', 'EPILEPSY', 'FENTANYL', 'AMATUER_ATHLETE',
       'GAMERS', 'GRADUATE_STUDENT', 'HEROIN', 'HIGH_IQ', 'LOW_IQ', 'KETAMINE',
       'INSOMNIA', 'LSD', 'MARIJUANA', 'METHAMPHETAMINE', 'MIGRAINE',
       'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN', 'ULTRASOMNIA',
       'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS', 'OXYCODONE',
       'PARKINSON'S_DISEASE', 'PERSCRIPTION_ANTIDEPRESSANTS',
       'PERSCRIPTION_SLEEPAIDS', 'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN',
       'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS', 'SAVANT_1', 'SAVANT_2',
       'SCHIZOPHRENIA', 'SELF_EMPLOYED', 'SINGLE_WORKING_PARENT',
       'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1', 'TERMINAL_2',
       'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT', 'AMATUER_MUSICIAN',
       'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
```

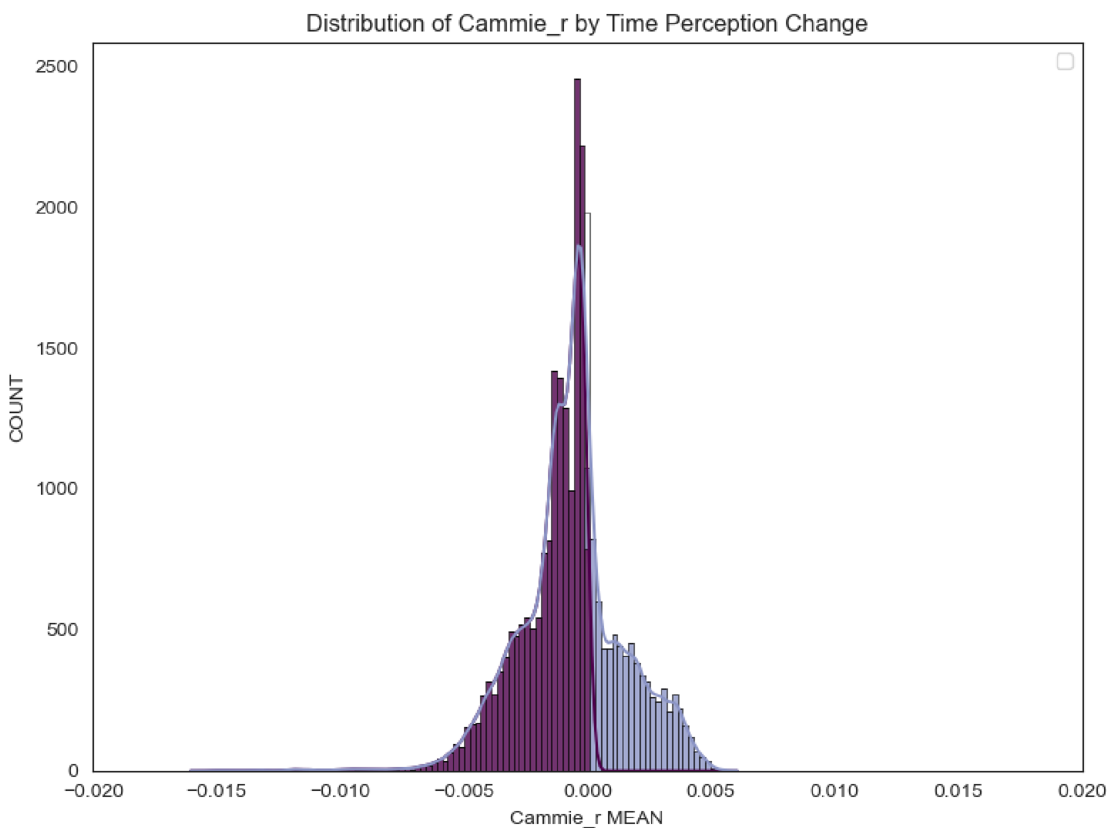
```
        'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
        'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
        'SDT_REFLECT', 'SDT_SLEEP_NUTRITION'],
       dtype='object')
```

[17]:

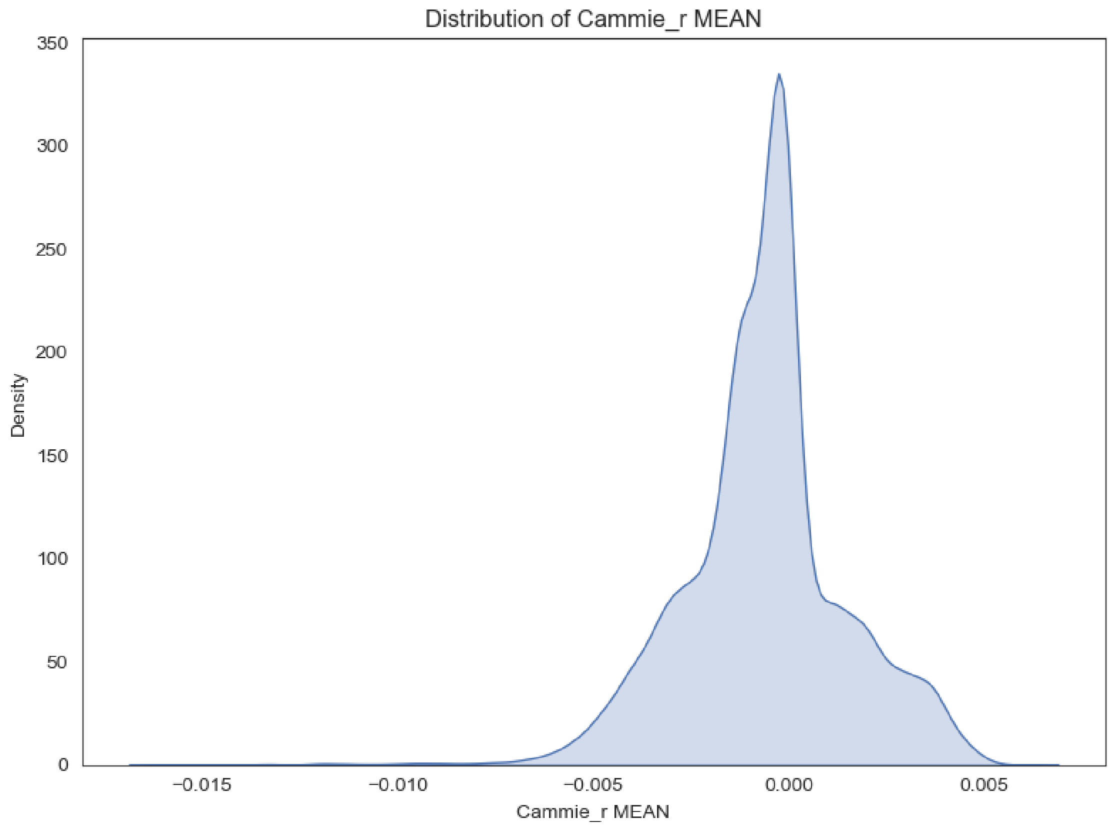
```
# Box Plot
target_names = {
    0: 'Average',
    1: 'Slower',
    2: 'Faster'}

# Now, use this mapping to replace the numerical values in your plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='TARGET', y='SEX', data=subset_df, palette='BuPu')

# Replace x-tick labels
plt.xticks(ticks=range(len(target_names)), labels=target_names.values())

plt.xlabel('TARGET')
plt.ylabel('SEX')
plt.title('Sex Distribution by Target')
plt.tight_layout()
plt.show()
```



[18]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'subset_df' is your DataFrame and 'target_names' is your dictionary

# Create the histogram
plt.figure(figsize=(8, 6))
ax = sns.histplot(subset_df, x='MEAN', bins=100, kde=True, hue='TARGET',␣
↪palette='BuPu', multiple='stack', alpha=0.8, edgecolor='black')

# Set labels, title, and axis limits
plt.xlabel('Cammie_r MEAN')
plt.ylabel('COUNT')
plt.title('Distribution of Cammie_r by Time Perception Change')
plt.xlim(-0.02, .02)
plt.grid(axis='y', alpha=0.001)

# Update legend with target names
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, [target_names[int(label)] for label in labels])

plt.tight_layout()
plt.show()
```



[19]:
```
plt.figure(figsize=(8, 6))
sns.kdeplot(subset_df['MEAN'], shade=True, color=(0.298, 0.447, 0.690))
plt.xlabel('Cammie_r MEAN')
plt.ylabel('Density')
plt.title('Distribution of Cammie_r MEAN')
plt.tight_layout()
plt.show()
```

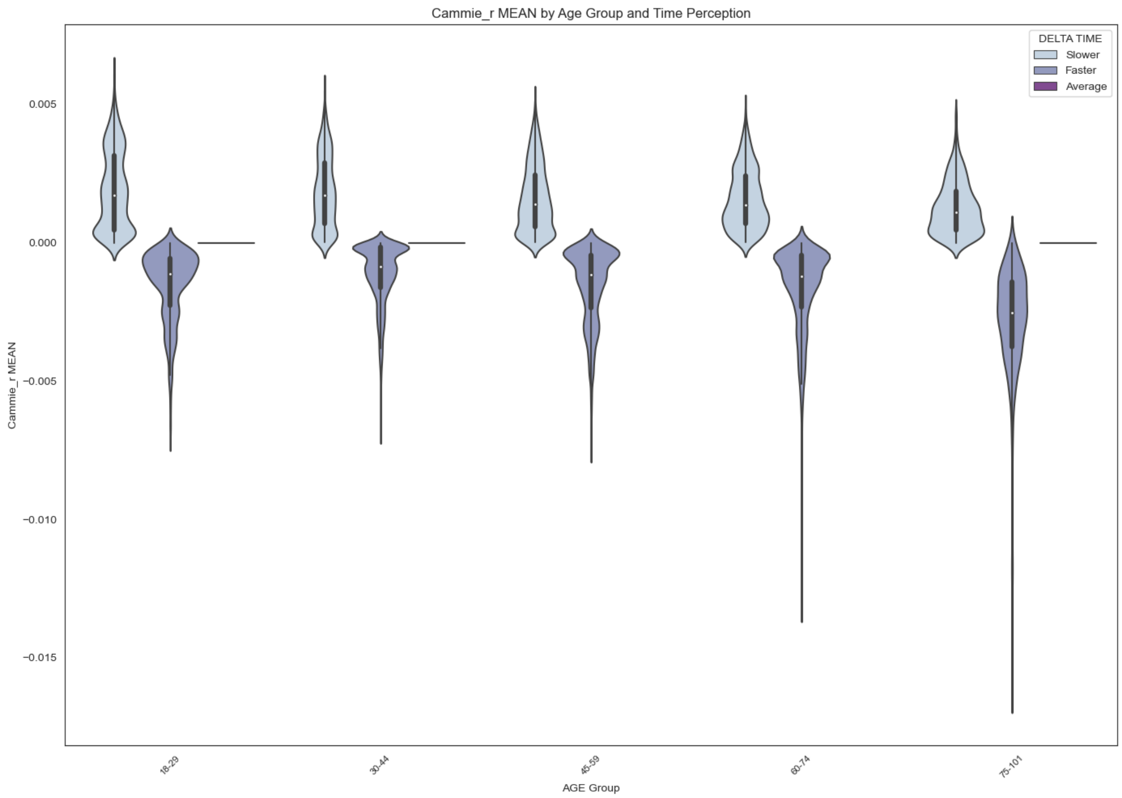


[20]:

```python
# Print basic statistics of 'MEAN'
print(subset_df['AGE'].describe())

# Print a few quantiles
print(subset_df['AGE'].quantile([0, 0.25, 0.5, 0.75, 1]))
```

```
count    26156.000000
mean        48.114161
std         18.817715
min         18.000000
25%         32.000000
50%         47.000000
75%         63.000000
max        100.000000
Name: AGE, dtype: float64
0.00     18.0
0.25     32.0
0.50     47.0
0.75     63.0
1.00    100.0
Name: AGE, dtype: float64
```

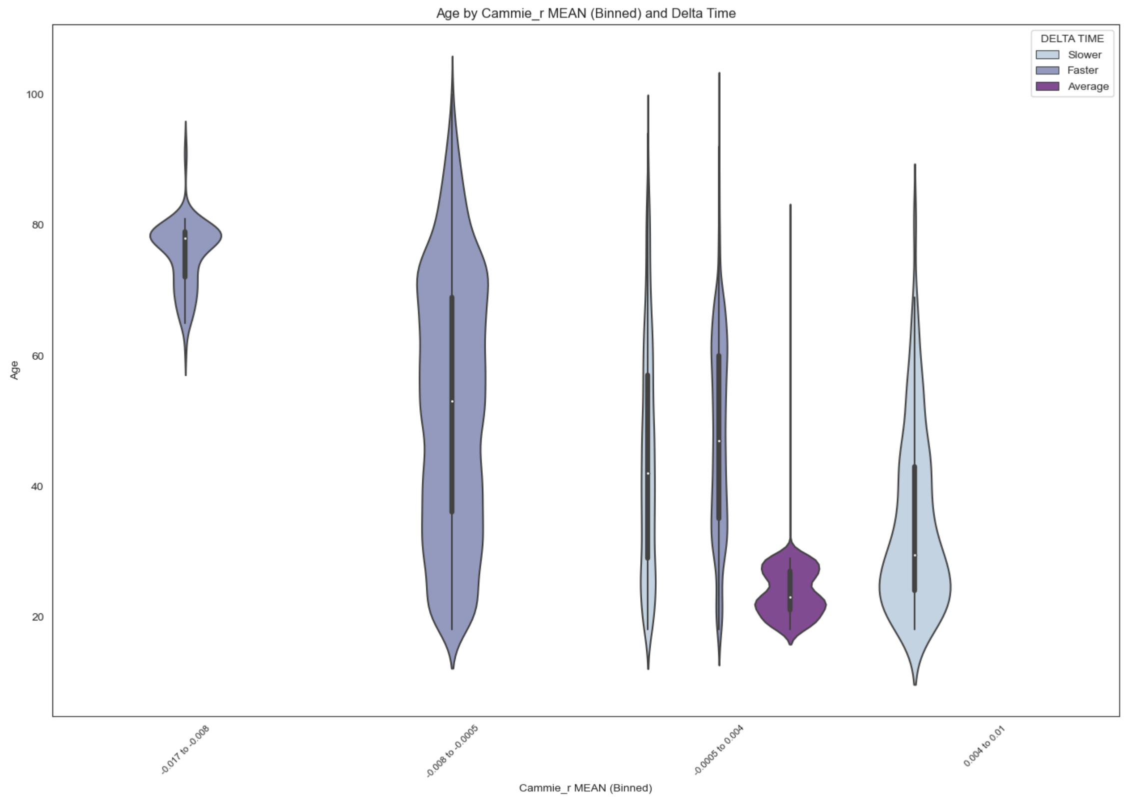
[21]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

target_names = {
    0: 'Average',
    1: 'Slower',
    2: 'Faster'}

# Map 'TARGET' to its corresponding names
subset_df['DELTA TIME'] = subset_df['TARGET'].map(target_names)

# Bin the 'AGE' data into categories
age_bins = [18, 29, 44, 59, 74, 101]
age_labels = ['18-29', '30-44', '45-59', '60-74', '75-101']
subset_df['AGE Group'] = pd.cut(subset_df['AGE'], bins=age_bins,␣
 ↪labels=age_labels, right=False)

# Create the violin plot using the binned age data
plt.figure(figsize=(14, 10))
sns.violinplot(x='AGE Group', y='MEAN', hue='DELTA TIME', data=subset_df,␣
 ↪palette='BuPu')

# Set labels and title
plt.xlabel('AGE Group')
plt.ylabel('Cammie_r MEAN')
plt.title('Cammie_r MEAN by Age Group and Time Perception')

# Rotate the x-tick labels
plt.xticks(rotation=45, fontsize='small')

# Tight layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()
```



[22]:
```
# Print basic statistics of 'MEAN'
print(subset_df['MEAN'].describe())

# Print a few quantiles
print(subset_df['MEAN'].quantile([0, 0.25, 0.5, 0.75, 1]))
```

```
count    26156.000000
mean        -0.000640
std          0.002101
min         -0.016030
25%         -0.001650
50%         -0.000460
75%          0.000210
max          0.006070
Name: MEAN, dtype: float64
0.00   -0.01603
0.25   -0.00165
0.50   -0.00046
0.75    0.00021
1.00    0.00607
Name: MEAN, dtype: float64
```

[23]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming 'subset_df' is your DataFrame and 'target_names' is your dictionary

# Map 'TARGET' to its corresponding names
subset_df['DELTA TIME'] = subset_df['TARGET'].map(target_names)

# Bin the 'MEAN' data into categories
bins = [-0.017, -0.008, -0.0005, 0.004, 0.007]
labels = ['-0.017 to -0.008', '-0.008 to -0.0005', '-0.0005 to 0.004', '0.004␣
↪to 0.01']
subset_df['MEAN Bins'] = pd.cut(subset_df['MEAN'], bins=bins, labels=labels)

# Create the violin plot using the binned data
plt.figure(figsize=(14, 10))
sns.violinplot(x='MEAN Bins', y='AGE', hue='DELTA TIME', data=subset_df,␣
↪palette='BuPu')

# Set labels and title
plt.xlabel('Cammie_r MEAN (Binned)')
plt.ylabel('Age')
plt.title('Age by Cammie_r MEAN (Binned) and Delta Time')

# Rotate the x-tick labels
plt.xticks(rotation=45, fontsize='small')

# Tight layout for better spacing
plt.tight_layout()

# Show the plot
plt.show()
```



10 NETWORK GRAPH

[24]:
```python
# Convert 'AGE' to integer
subset_df['AGE'] = subset_df['AGE'].astype(int)

# Round 'MEAN' to six decimal places and ensure it's float
subset_df['MEAN'] = subset_df['MEAN'].round(6).astype(float)

# Check the data types and a sample of the data
print(subset_df.dtypes)
print(subset_df.head())
```

```
AGE                        int32
SEX                       object
TARGET                     int64
MEAN                     float64
ADHD                     float64
                           …
SDT_REFLECT              float64
SDT_SLEEP_NUTRITION      float64
DELTA TIME                object
AGE Group               category
```

```
MEAN Bins               category
Length: 80, dtype: object
   AGE SEX  TARGET     MEAN    ADHD  30_49  50_69  70_89  90+  ALCOHOL  …  \
0   18   M       1  0.00034 -0.0604    0.0    0.0    0.0  0.0      0.0  …
1   18   M       2 -0.00201 -0.0604    0.0    0.0    0.0  0.0      0.0  …
2   18   M       2 -0.00039 -0.0604    0.0    0.0    0.0  0.0      0.0  …
3   18   M       1  0.00159 -0.0604    0.0    0.0    0.0  0.0      0.0  …
4   18   M       1  0.00136 -0.0604    0.0    0.0    0.0  0.0      0.0  …

   SDT_NEW_EXPERIENCES  SDT_VARY_ACTIVITIES  SDT_REDUCE_STRESS  \
0               0.0000               0.0667              0.000
1               0.0000               0.0667              0.000
2               0.0000               0.0667              0.000
3               0.0708               0.0667              0.125
4               0.0708               0.0667              0.125

   SDT_LIMIT_MULTITASKING  SDT_CREATIVE_PURSUITS  SDT_REFLECT  \
0                     0.0                    0.0          0.0
1                     0.0                    0.0          0.0
2                     0.0                    0.0          0.0
3                     0.0                    0.0          0.0
4                     0.0                    0.0          0.0

   SDT_SLEEP_NUTRITION  DELTA TIME  AGE Group          MEAN Bins
0               0.1563      Slower      18-29   -0.0005 to 0.004
1               0.1563      Faster      18-29  -0.008 to -0.0005
2               0.1563      Faster      18-29   -0.0005 to 0.004
3               0.1563      Slower      18-29   -0.0005 to 0.004
4               0.1563      Slower      18-29   -0.0005 to 0.004

[5 rows x 80 columns]
```

[25]:
```python
# Identify NaN values
nan_counts = subset_df.isna().sum()
print("NaN counts before filling:")
print(nan_counts)

# Calculate the mean of the 'AGE' column, excluding NaNs
age_mean = subset_df['AGE'].mean(skipna=True)

# Fill NaN values in the 'AGE' column with the calculated mean
subset_df['AGE'].fillna(value=age_mean, inplace=True)

# Verify the operation
print(subset_df['AGE'])

# Fill NaN values in the 'MEAN' column with 0
```

```python
subset_df['MEAN'].fillna(0, inplace=True)

# Verify the operation
print(subset_df['MEAN'])

# Check again for NaN values
nan_counts_after = subset_df.isna().sum()
print("\nNaN counts after filling:")
print(nan_counts_after)
```

```
NaN counts before filling:
AGE                    0
SEX                    0
TARGET                 0
MEAN                   0
ADHD                   0
                      ..
SDT_REFLECT            0
SDT_SLEEP_NUTRITION    0
DELTA TIME             0
AGE Group              0
MEAN Bins              0
Length: 80, dtype: int64
0          18
1          18
2          18
3          18
4          18
         …
26151     100
26152     100
26153     100
26154     100
26155     100
Name: AGE, Length: 26156, dtype: int32
0         0.00034
1        -0.00201
2        -0.00039
3         0.00159
4         0.00136
           …
26151    -0.00756
26152    -0.00421
26153    -0.00394
26154    -0.00358
26155    -0.00320
Name: MEAN, Length: 26156, dtype: float64
```

```
NaN counts after filling:
AGE                     0
SEX                     0
TARGET                  0
MEAN                    0
ADHD                    0
                       ..
SDT_REFLECT             0
SDT_SLEEP_NUTRITION     0
DELTA TIME              0
AGE Group               0
MEAN Bins               0
Length: 80, dtype: int64
```

[26]:
```
# Find the indices of rows where NaN values are present
nan_rows = subset_df[subset_df.isna().any(axis=1)]

# Print the row indices where NaN values are found
print(nan_rows.index.tolist())
```

```
[]
```

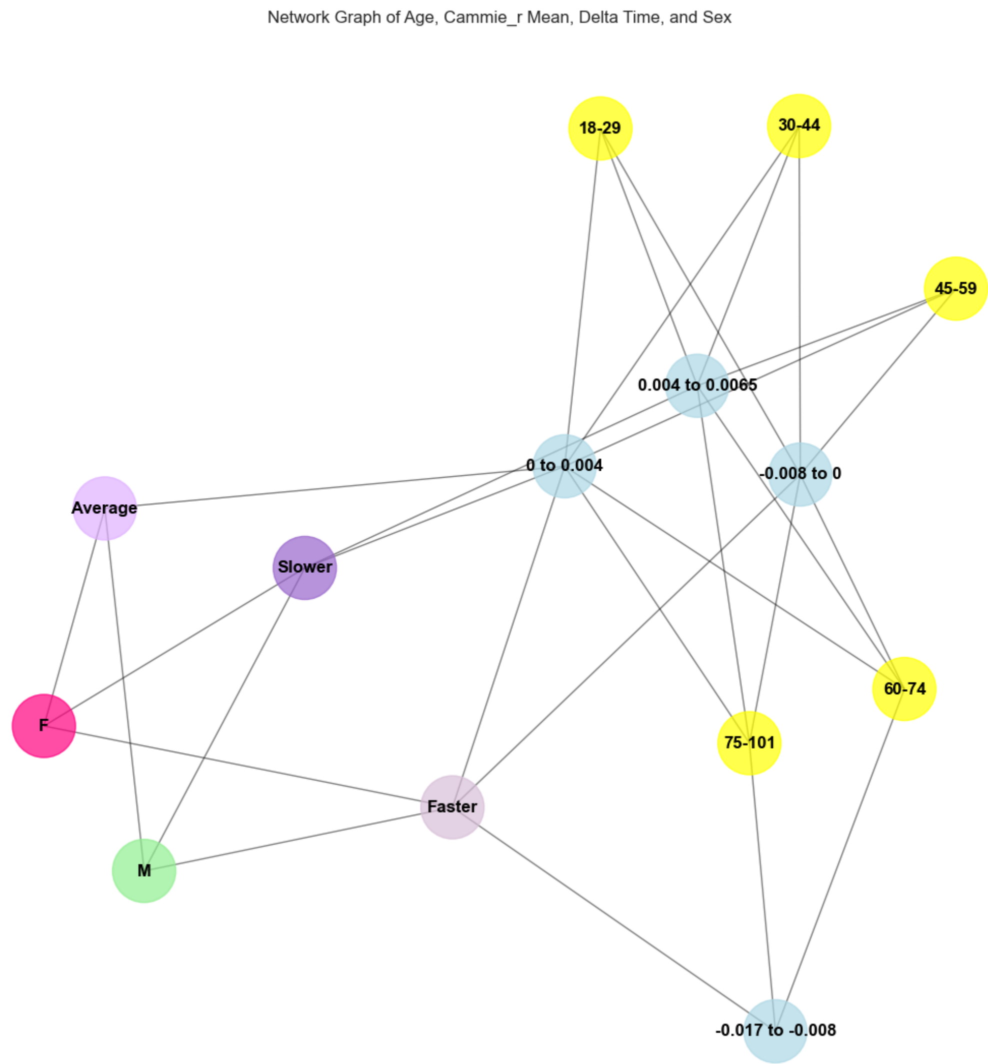
[27]:
```
# Create a dictionary to store the column names and their corresponding row␣
 ↪indices with NaN values
nan_details = {}

# Iterate over each column and find rows with NaN values
for column in subset_df.columns:
    nan_rows = subset_df[subset_df[column].isna()]
    if not nan_rows.empty:
        nan_details[column] = nan_rows.index.tolist()

# Print the details
for column, rows in nan_details.items():
    print(f"Column '{column}' has NaN values at rows: {rows}")
```

[28]:
```
subset_df.to_excel('subset_df_file.xlsx', index=False)
```

[29]:
```
import networkx as nx

# Bin 'AGE' and 'MEAN', and map 'TARGET'
age_bins = [18, 29, 44, 59, 74, 101]
mean_bins = [-0.017, -0.008, 0, 0.004, 0.0065]
age_labels = ['18-29', '30-44', '45-59', '60-74', '75-101']
mean_labels = ['-0.017 to -0.008', '-0.008 to 0', '0 to 0.004', '0.004 to 0.
 ↪0065']
```

```
subset_df['AGE Group'] = pd.cut(subset_df['AGE'], bins=age_bins,␣
↪labels=age_labels, right=False)
subset_df['MEAN Group'] = pd.cut(subset_df['MEAN'], bins=mean_bins,␣
↪labels=mean_labels, right=False)
subset_df['TARGET Name'] = subset_df['TARGET'].map(target_names)

# Create a graph
G = nx.Graph()

# Add nodes for each group
for group in age_labels + mean_labels + list(target_names.values()) +␣
↪subset_df['SEX'].unique().tolist():
    G.add_node(group, type='group')

# Add edges
for _, row in subset_df.iterrows():
    G.add_edge(row['AGE Group'], row['MEAN Group'])
    G.add_edge(row['MEAN Group'], row['TARGET Name'])
    G.add_edge(row['TARGET Name'], row['SEX'])  # Adding edge for 'SEX'

# Define node colors
def get_node_color(node, node_data):
    if node_data['type'] == 'group':
        if node in age_labels:
            return 'yellow'  # Color for 'AGE Group'
        elif node in mean_labels:
            return 'lightblue'  # Color for 'MEAN Group'
        elif node == 'Average':
            return '#E0B0FF'  # Hexadecimal for dark purple
        elif node == 'Faster':
            return '#D8BFD8'
        elif node == 'Slower':
            return '#9966CC'
        elif node == 'F':
            return '#FF007F'  # Color for Female
        elif node == 'M':
            return 'lightgreen'  # Color for Male
    return 'darkblue'  # Default color

# Draw the graph
plt.figure(figsize=(14, 14))
pos = nx.spring_layout(G)  # positions for all nodes

# Customize node size, node color, and node labels
node_colors = [get_node_color(node, data) for node, data in G.nodes(data=True)]
node_sizes = [3000 if node in ['target_names'] else 2000 for node in G.nodes()]
```

```
nx.draw_networkx_nodes(G, pos, node_color=node_colors, node_size=node_sizes,␣
↪alpha=0.7)

# Customize edge width and transparency
nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)

# Customize node labels (bold, white text for 'M' and 'TARGET Name')
labels = {}
for node, data in G.nodes(data=True):
    label_color = 'white' if node in ['M', 'TARGET Name'] else 'black'
    labels[node] = node
nx.draw_networkx_labels(G, pos, labels=labels, font_size=12,␣
↪font_weight='bold', font_color=label_color)

plt.title("Network Graph of Age, Cammie_r Mean, Delta Time, and Sex")
plt.axis('off')  # Turn off axis
plt.show()

plt.figure(figsize=(14, 14))
# Your code to create and customize the network graph here

# Save the graph as an image
plt.savefig('network_graph.png', format='png', dpi=300)
```



<Figure size 1400x1400 with 0 Axes>

11 Build, Test and Visualize the Model Performance

[30]:
```
column_names = subset_df.columns.tolist()
print(column_names)

model_subset_df = subset_df.drop(columns=['DELTA TIME', 'SEX', 'AGE Group',␣
↪'MEAN Bins', 'MEAN Group', 'TARGET Name'])
```

```
['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '30_49', '50_69', '70_89', '90+',
'ALCOHOL', "ALZHEIMER'S", 'ANXIETY', 'AUTISM_1', 'AUTISM_2', 'BINGE_WATCHERS',
'BIPOLAR_I', 'BIPOLAR_II', 'CHRONIC_PAIN', 'COCAINE', 'C_MEDITATION',
```

```
'DEPRESSION', 'DISSOCIATIVE_DISORDER', 'PROFESSIONAL_ATHLETE', 'EPILEPSY',
'FENTANYL', 'AMATUER_ATHLETE', 'GAMERS', 'GRADUATE_STUDENT', 'HEROIN',
'HIGH_IQ', 'LOW_IQ', 'KETAMINE', 'INSOMNIA', 'LSD', 'MARIJUANA',
'METHAMPHETAMINE', 'MIGRAINE', 'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN',
'ULTRASOMNIA', 'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS', 'OXYCODONE',
"PARKINSON'S_DISEASE", 'PERSCRIPTION_ANTIDEPRESSANTS', 'PERSCRIPTION_SLEEPAIDS',
'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN', 'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS',
'SAVANT_1', 'SAVANT_2', 'SCHIZOPHRENIA', 'SELF_EMPLOYED',
'SINGLE_WORKING_PARENT', 'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1',
'TERMINAL_2', 'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT',
'AMATUER_MUSICIAN', 'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
'SDT_REFLECT', 'SDT_SLEEP_NUTRITION', 'DELTA TIME', 'AGE Group', 'MEAN Bins',
'MEAN Group', 'TARGET Name']
```

[31]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import shap
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

# Ensure using TensorFlow 2.x
print("TensorFlow version:", tf.__version__)

# Load your dataset
# model_subset_df = pd.read_csv('your_dataset.csv')

# Separate the input features (X) and the target label (y)
X = model_subset_df.drop(columns=['TARGET'])
y = model_subset_df['TARGET']

# Split the data into train, test, and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪15, random_state=42)

# Perform any necessary preprocessing
scaler = StandardScaler()
```

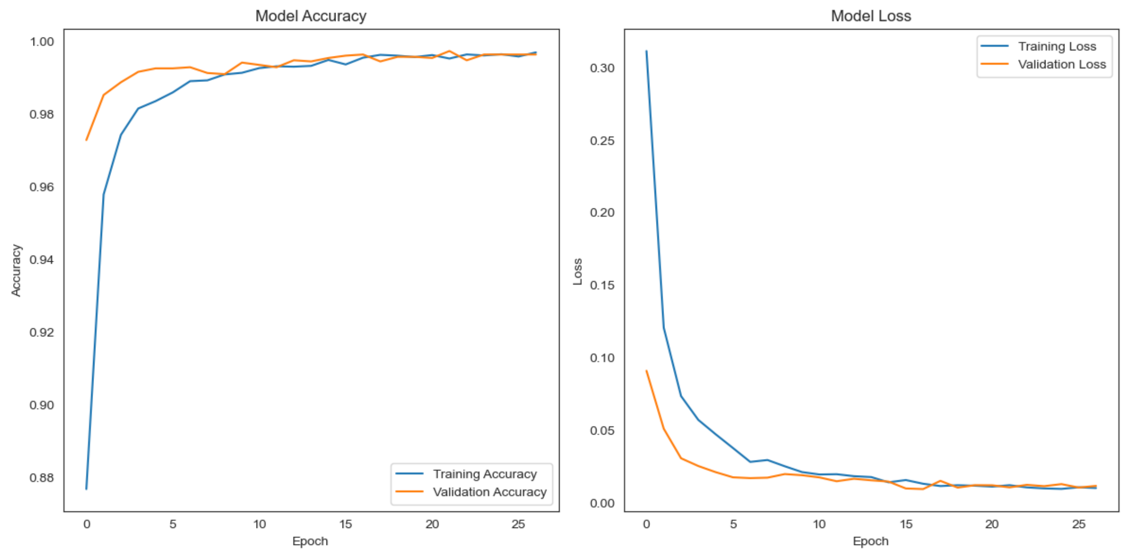
```python
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# One-hot encode the target labels
y_train_encoded = to_categorical(y_train, num_classes=3)
y_val_encoded = to_categorical(y_val, num_classes=3)
y_test_encoded = to_categorical(y_test, num_classes=3)

# Build the deep learning model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Define the optimizer
optimizer = Adam(learning_rate=0.001)

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

# Define Early Stopping callback
early_stopping = EarlyStopping(patience=10, monitor='val_loss', restore_best_weights=True)

# Train the model with Early Stopping
history = model.fit(X_train_scaled, y_train_encoded, epochs=100, batch_size=32, validation_data=(X_val_scaled, y_val_encoded), callbacks=[early_stopping], verbose=2)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test_encoded)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)

# Calculate test accuracy as a percentage
test_accuracy_percentage = test_accuracy * 100
print('Test Accuracy (Percentage): {:.2f}%'.format(test_accuracy_percentage))

# Make predictions and get the predicted class labels
predicted_labels = [np.argmax(pred) for pred in model.predict(X_test_scaled)]

# Plot the training and validation accuracy and loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
```

```python
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# SHAP Analysis (Optional)
# Note: This section might require modifications based on your specific model␣
↪and data
def model_predict(data):
    return model.predict(scaler.transform(data))

explainer = shap.KernelExplainer(model_predict, shap.sample(X_train, 100))
shap_values = explainer.shap_values(shap.sample(X_test, 100))
shap.summary_plot(shap_values, shap.sample(X_test, 100))
```

```
TensorFlow version: 2.15.0
WARNING:tensorflow:From C:\Users\newmy\anaconda3\lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

Epoch 1/100
WARNING:tensorflow:From C:\Users\newmy\anaconda3\lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\newmy\anaconda3\lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

556/556 - 1s - loss: 0.3115 - accuracy: 0.8768 - val_loss: 0.0911 -
val_accuracy: 0.9729 - 1s/epoch - 2ms/step
Epoch 2/100
556/556 - 1s - loss: 0.1208 - accuracy: 0.9579 - val_loss: 0.0512 -
val_accuracy: 0.9853 - 513ms/epoch - 923us/step
```

```
Epoch 3/100
556/556 - 1s - loss: 0.0737 - accuracy: 0.9744 - val_loss: 0.0308 -
val_accuracy: 0.9888 - 501ms/epoch - 901us/step
Epoch 4/100
556/556 - 1s - loss: 0.0572 - accuracy: 0.9816 - val_loss: 0.0255 -
val_accuracy: 0.9917 - 501ms/epoch - 901us/step
Epoch 5/100
556/556 - 1s - loss: 0.0474 - accuracy: 0.9836 - val_loss: 0.0213 -
val_accuracy: 0.9927 - 511ms/epoch - 919us/step
Epoch 6/100
556/556 - 1s - loss: 0.0379 - accuracy: 0.9861 - val_loss: 0.0177 -
val_accuracy: 0.9927 - 514ms/epoch - 924us/step
Epoch 7/100
556/556 - 1s - loss: 0.0283 - accuracy: 0.9891 - val_loss: 0.0171 -
val_accuracy: 0.9930 - 509ms/epoch - 915us/step
Epoch 8/100
556/556 - 1s - loss: 0.0296 - accuracy: 0.9894 - val_loss: 0.0174 -
val_accuracy: 0.9914 - 525ms/epoch - 944us/step
Epoch 9/100
556/556 - 1s - loss: 0.0253 - accuracy: 0.9910 - val_loss: 0.0199 -
val_accuracy: 0.9911 - 502ms/epoch - 902us/step
Epoch 10/100
556/556 - 1s - loss: 0.0213 - accuracy: 0.9915 - val_loss: 0.0191 -
val_accuracy: 0.9943 - 508ms/epoch - 914us/step
Epoch 11/100
556/556 - 1s - loss: 0.0196 - accuracy: 0.9927 - val_loss: 0.0176 -
val_accuracy: 0.9936 - 511ms/epoch - 920us/step
Epoch 12/100
556/556 - 1s - loss: 0.0198 - accuracy: 0.9933 - val_loss: 0.0150 -
val_accuracy: 0.9930 - 504ms/epoch - 907us/step
Epoch 13/100
556/556 - 1s - loss: 0.0185 - accuracy: 0.9931 - val_loss: 0.0167 -
val_accuracy: 0.9949 - 526ms/epoch - 947us/step
Epoch 14/100
556/556 - 1s - loss: 0.0179 - accuracy: 0.9934 - val_loss: 0.0156 -
val_accuracy: 0.9946 - 507ms/epoch - 911us/step
Epoch 15/100
556/556 - 1s - loss: 0.0142 - accuracy: 0.9950 - val_loss: 0.0147 -
val_accuracy: 0.9955 - 506ms/epoch - 909us/step
Epoch 16/100
556/556 - 1s - loss: 0.0158 - accuracy: 0.9938 - val_loss: 0.0100 -
val_accuracy: 0.9962 - 511ms/epoch - 919us/step
Epoch 17/100
556/556 - 1s - loss: 0.0132 - accuracy: 0.9956 - val_loss: 0.0096 -
val_accuracy: 0.9965 - 510ms/epoch - 917us/step
Epoch 18/100
556/556 - 1s - loss: 0.0117 - accuracy: 0.9964 - val_loss: 0.0152 -
val_accuracy: 0.9946 - 505ms/epoch - 908us/step
```

```
Epoch 19/100
556/556 - 1s - loss: 0.0122 - accuracy: 0.9962 - val_loss: 0.0106 -
val_accuracy: 0.9959 - 505ms/epoch - 909us/step
Epoch 20/100
556/556 - 1s - loss: 0.0119 - accuracy: 0.9958 - val_loss: 0.0123 -
val_accuracy: 0.9959 - 503ms/epoch - 905us/step
Epoch 21/100
556/556 - 1s - loss: 0.0113 - accuracy: 0.9963 - val_loss: 0.0122 -
val_accuracy: 0.9955 - 502ms/epoch - 903us/step
Epoch 22/100
556/556 - 1s - loss: 0.0123 - accuracy: 0.9954 - val_loss: 0.0108 -
val_accuracy: 0.9975 - 504ms/epoch - 907us/step
Epoch 23/100
556/556 - 0s - loss: 0.0107 - accuracy: 0.9965 - val_loss: 0.0125 -
val_accuracy: 0.9949 - 499ms/epoch - 897us/step
Epoch 24/100
556/556 - 1s - loss: 0.0100 - accuracy: 0.9962 - val_loss: 0.0116 -
val_accuracy: 0.9965 - 502ms/epoch - 903us/step
Epoch 25/100
556/556 - 0s - loss: 0.0097 - accuracy: 0.9966 - val_loss: 0.0130 -
val_accuracy: 0.9965 - 496ms/epoch - 892us/step
Epoch 26/100
556/556 - 1s - loss: 0.0108 - accuracy: 0.9960 - val_loss: 0.0106 -
val_accuracy: 0.9965 - 501ms/epoch - 901us/step
Epoch 27/100
556/556 - 1s - loss: 0.0103 - accuracy: 0.9971 - val_loss: 0.0118 -
val_accuracy: 0.9965 - 504ms/epoch - 906us/step
164/164 [==============================] - 0s 709us/step - loss: 0.0124 -
accuracy: 0.9952
Test Loss: 0.012394802644848824
Test Accuracy: 0.995221734046936
Test Accuracy (Percentage): 99.52%
164/164 [==============================] - 0s 660us/step
```




```
4/4 [==============================] - 0s 2ms/step

  0%|
| 0/100 [00:00<?, ?it/s]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 4s 656us/step

  1%|
| 1/100 [00:06<10:20,  6.27s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 654us/step

  2%|
| 2/100 [00:12<09:54,  6.06s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 657us/step

  3%|
| 3/100 [00:18<09:42,  6.01s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 668us/step

  4%|
| 4/100 [00:24<09:47,  6.12s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 4s 655us/step

  5%|
| 5/100 [00:30<09:36,  6.06s/it]
```

```
1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 653us/step

  6%|
| 6/100 [00:36<09:25,  6.01s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 653us/step

  7%|
| 7/100 [00:42<09:15,  5.97s/it]

1/1 [==============================] - 0s 13ms/step
6800/6800 [==============================] - 4s 658us/step

  8%|
| 8/100 [00:48<09:18,  6.07s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 4s 657us/step

  9%|
| 9/100 [00:54<09:08,  6.02s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 654us/step

 10%|
| 10/100 [01:00<08:58,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 655us/step

 11%|
| 11/100 [01:06<08:51,  5.97s/it]

1/1 [==============================] - 0s 14ms/step
6794/6794 [==============================] - 4s 658us/step

 12%|
| 12/100 [01:12<08:52,  6.05s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 665us/step

 13%|
| 13/100 [01:18<08:44,  6.03s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 659us/step

 14%|
| 14/100 [01:24<08:35,  6.00s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 667us/step
```

```
 15%|
| 15/100 [01:30<08:29,  6.00s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 666us/step

 16%|
| 16/100 [01:36<08:23,  6.00s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 4s 656us/step

 17%|
| 17/100 [01:42<08:16,  5.98s/it]

1/1 [==============================] - 0s 16ms/step
6788/6788 [==============================] - 4s 654us/step

 18%|
| 18/100 [01:48<08:11,  5.99s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 4s 654us/step

 19%|
| 19/100 [01:54<08:03,  5.97s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 651us/step

 20%|
| 20/100 [02:00<07:55,  5.95s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 661us/step

 21%|
| 21/100 [02:06<07:50,  5.96s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 691us/step

 22%|
| 22/100 [02:12<07:49,  6.02s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 674us/step

 23%|
| 23/100 [02:18<07:44,  6.04s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 661us/step

 24%|
| 24/100 [02:24<07:37,  6.02s/it]
```

```
1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 659us/step

 25%|
| 25/100 [02:30<07:29,  6.00s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 659us/step

 26%|
| 26/100 [02:36<07:22,  5.98s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 657us/step

 27%|
| 27/100 [02:42<07:15,  5.97s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 656us/step

 28%|
| 28/100 [02:48<07:08,  5.95s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 661us/step

 29%|
| 29/100 [02:54<07:02,  5.95s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 653us/step

 30%|
| 30/100 [02:59<06:56,  5.95s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 649us/step

 31%|
| 31/100 [03:05<06:48,  5.93s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 657us/step

 32%|
| 32/100 [03:11<06:42,  5.92s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 661us/step

 33%|
| 33/100 [03:17<06:37,  5.94s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 656us/step
```

```
 34%|
| 34/100 [03:23<06:31,  5.93s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 653us/step

 35%|
| 35/100 [03:29<06:25,  5.93s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 657us/step

 36%|
| 36/100 [03:35<06:19,  5.93s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 662us/step

 37%|
| 37/100 [03:41<06:14,  5.94s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 673us/step

 38%|
| 38/100 [03:47<06:10,  5.98s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 666us/step

 39%|
| 39/100 [03:53<06:05,  5.99s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 662us/step

 40%|
| 40/100 [03:59<05:59,  5.98s/it]

1/1 [==============================] - 0s 12ms/step
6794/6794 [==============================] - 4s 658us/step

 41%|
| 41/100 [04:05<05:52,  5.97s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 4s 657us/step

 42%|
| 42/100 [04:11<05:45,  5.96s/it]

1/1 [==============================] - 0s 14ms/step
6794/6794 [==============================] - 4s 659us/step

 43%|
| 43/100 [04:17<05:39,  5.96s/it]
```

```
1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 664us/step

 44%|
| 44/100 [04:23<05:33,  5.96s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 657us/step

 45%|
| 45/100 [04:29<05:27,  5.96s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 662us/step

 46%|
| 46/100 [04:35<05:22,  5.96s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 661us/step

 47%|
| 47/100 [04:41<05:16,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 656us/step

 48%|
| 48/100 [04:47<05:10,  5.96s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 666us/step

 49%|
| 49/100 [04:53<05:04,  5.97s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 663us/step

 50%|
| 50/100 [04:59<04:59,  5.99s/it]

1/1 [==============================] - 0s 20ms/step
6788/6788 [==============================] - 4s 656us/step

 51%|
| 51/100 [05:05<04:52,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 670us/step

 52%|
| 52/100 [05:11<04:47,  5.99s/it]

1/1 [==============================] - 0s 16ms/step
6788/6788 [==============================] - 5s 662us/step
```

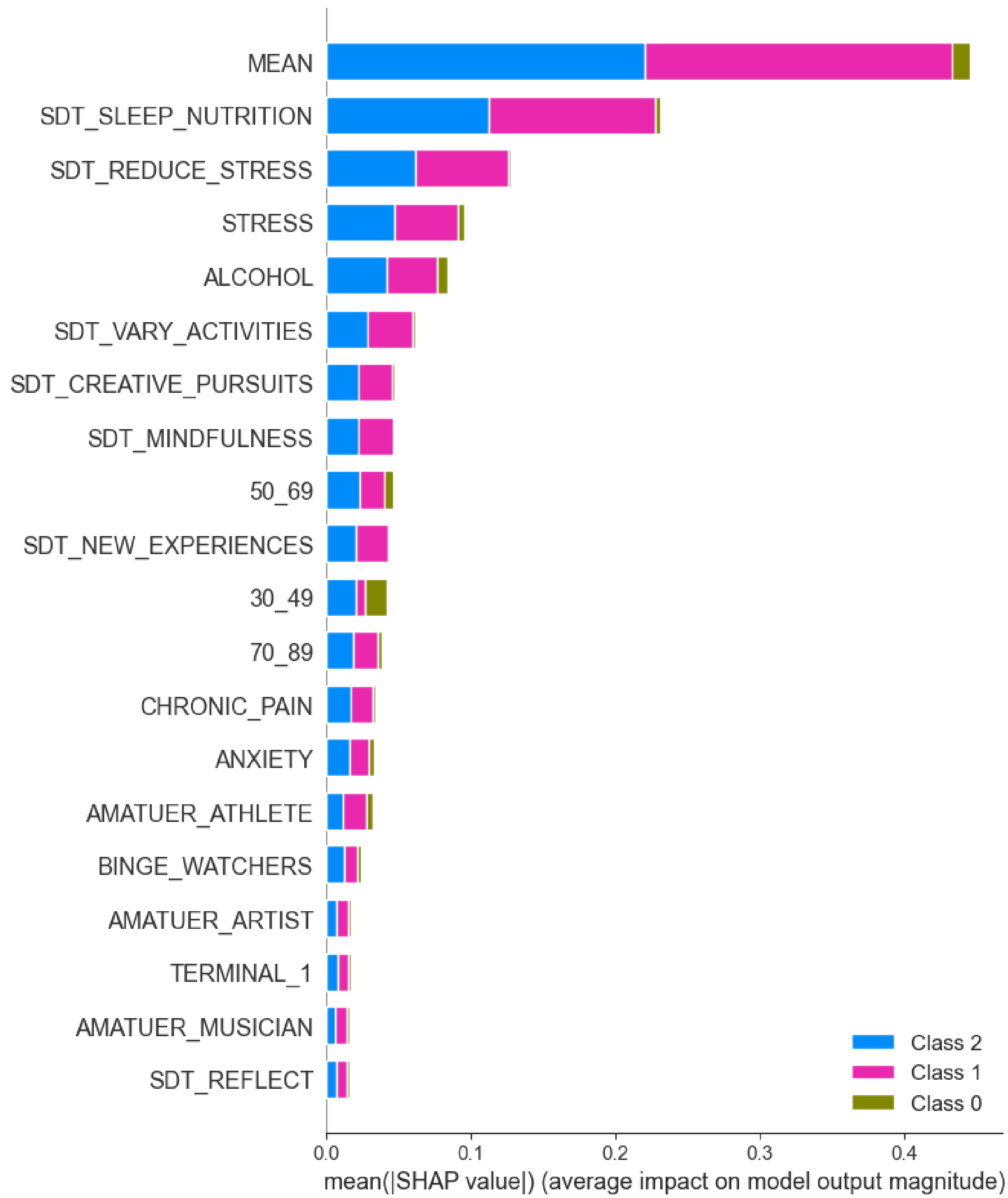
```
 53%|
| 53/100 [05:17<04:41,  5.99s/it]

1/1 [==============================] - 0s 20ms/step
6794/6794 [==============================] - 4s 660us/step

 54%|
| 54/100 [05:23<04:35,  5.99s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 4s 661us/step

 55%|
| 55/100 [05:29<04:29,  5.98s/it]

1/1 [==============================] - 0s 16ms/step
6788/6788 [==============================] - 5s 661us/step

 56%|
| 56/100 [05:35<04:22,  5.97s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 5s 661us/step

 57%|
| 57/100 [05:41<04:16,  5.97s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 4s 659us/step

 58%|
| 58/100 [05:47<04:19,  6.19s/it]

1/1 [==============================] - 0s 18ms/step
6788/6788 [==============================] - 5s 670us/step

 59%|
| 59/100 [05:53<04:11,  6.14s/it]

1/1 [==============================] - 0s 20ms/step
6788/6788 [==============================] - 4s 659us/step

 60%|
| 60/100 [05:59<04:03,  6.08s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 662us/step

 61%|
| 61/100 [06:05<03:56,  6.06s/it]

1/1 [==============================] - 0s 10ms/step
6794/6794 [==============================] - 5s 662us/step

 62%|
| 62/100 [06:11<03:49,  6.04s/it]
```

```
1/1 [==============================] - 0s 9ms/step
6788/6788 [==============================] - 5s 682us/step

 63%|
| 63/100 [06:17<03:44,  6.07s/it]

1/1 [==============================] - 0s 8ms/step
6788/6788 [==============================] - 5s 661us/step

 64%|
| 64/100 [06:23<03:37,  6.04s/it]

1/1 [==============================] - 0s 16ms/step
6788/6788 [==============================] - 4s 657us/step

 65%|
| 65/100 [06:29<03:30,  6.01s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 4s 660us/step

 66%|
| 66/100 [06:35<03:23,  5.99s/it]

1/1 [==============================] - 0s 9ms/step
6788/6788 [==============================] - 4s 657us/step

 67%|
| 67/100 [06:41<03:16,  5.97s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 663us/step

 68%|
| 68/100 [06:47<03:11,  5.97s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 5s 669us/step

 69%|
| 69/100 [06:53<03:05,  5.99s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 662us/step

 70%|
| 70/100 [06:59<02:59,  6.00s/it]

1/1 [==============================] - 0s 12ms/step
6800/6800 [==============================] - 4s 655us/step

 71%|
| 71/100 [07:05<02:53,  5.98s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 659us/step
```

```
 72%|
| 72/100 [07:11<02:47,  5.97s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 5s 662us/step

 73%|
| 73/100 [07:17<02:41,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 667us/step

 74%|
| 74/100 [07:23<02:35,  5.99s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 5s 672us/step

 75%|
| 75/100 [07:29<02:30,  6.01s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 665us/step

 76%|
| 76/100 [07:35<02:24,  6.00s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 682us/step

 77%|
| 77/100 [07:41<02:19,  6.05s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 663us/step

 78%|
| 78/100 [07:47<02:12,  6.03s/it]

1/1 [==============================] - 0s 8ms/step
6794/6794 [==============================] - 5s 664us/step

 79%|
| 79/100 [07:53<02:06,  6.02s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 4s 658us/step

 80%|
| 80/100 [07:59<02:01,  6.05s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 666us/step

 81%|
| 81/100 [08:05<01:54,  6.03s/it]
```

```
1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 4s 659us/step

 82%|
| 82/100 [08:11<01:48,  6.01s/it]

1/1 [==============================] - 0s 14ms/step
6788/6788 [==============================] - 5s 662us/step

 83%|
| 83/100 [08:17<01:41,  6.00s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 661us/step

 84%|
| 84/100 [08:23<01:35,  5.99s/it]

1/1 [==============================] - 0s 8ms/step
6788/6788 [==============================] - 5s 669us/step

 85%|
| 85/100 [08:29<01:29,  5.99s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 4s 658us/step

 86%|
| 86/100 [08:35<01:23,  5.97s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 5s 662us/step

 87%|
| 87/100 [08:41<01:17,  5.96s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 662us/step

 88%|
| 88/100 [08:47<01:11,  5.96s/it]

1/1 [==============================] - 0s 12ms/step
6788/6788 [==============================] - 5s 665us/step

 89%|
| 89/100 [08:53<01:05,  5.97s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 664us/step

 90%|
| 90/100 [08:59<00:59,  5.97s/it]

1/1 [==============================] - 0s 10ms/step
6788/6788 [==============================] - 4s 659us/step
```

```
 91%|
| 91/100 [09:05<00:53,  5.96s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 4s 658us/step

 92%|
| 92/100 [09:11<00:47,  5.95s/it]

1/1 [==============================] - 0s 15ms/step
6788/6788 [==============================] - 5s 664us/step

93%|
| 93/100 [09:17<00:41,  5.96s/it]

1/1 [==============================] - 0s 17ms/step
6794/6794 [==============================] - 5s 669us/step

 94%|
     | 94/100 [09:23<00:35,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6794/6794 [==============================] - 4s 657us/step

 95%|
     | 95/100 [09:29<00:29,  5.97s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 663us/step

 96%|
    | 96/100 [09:35<00:24,  6.01s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 653us/step

 97%|
    | 97/100 [09:41<00:17,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 662us/step

 98%|
   | 98/100 [09:47<00:11,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 5s 666us/step

 99%|
   | 99/100 [09:53<00:05,  5.98s/it]

1/1 [==============================] - 0s 13ms/step
6788/6788 [==============================] - 4s 660us/step

100%|
  | 100/100 [09:59<00:00,  5.99s/it]
```

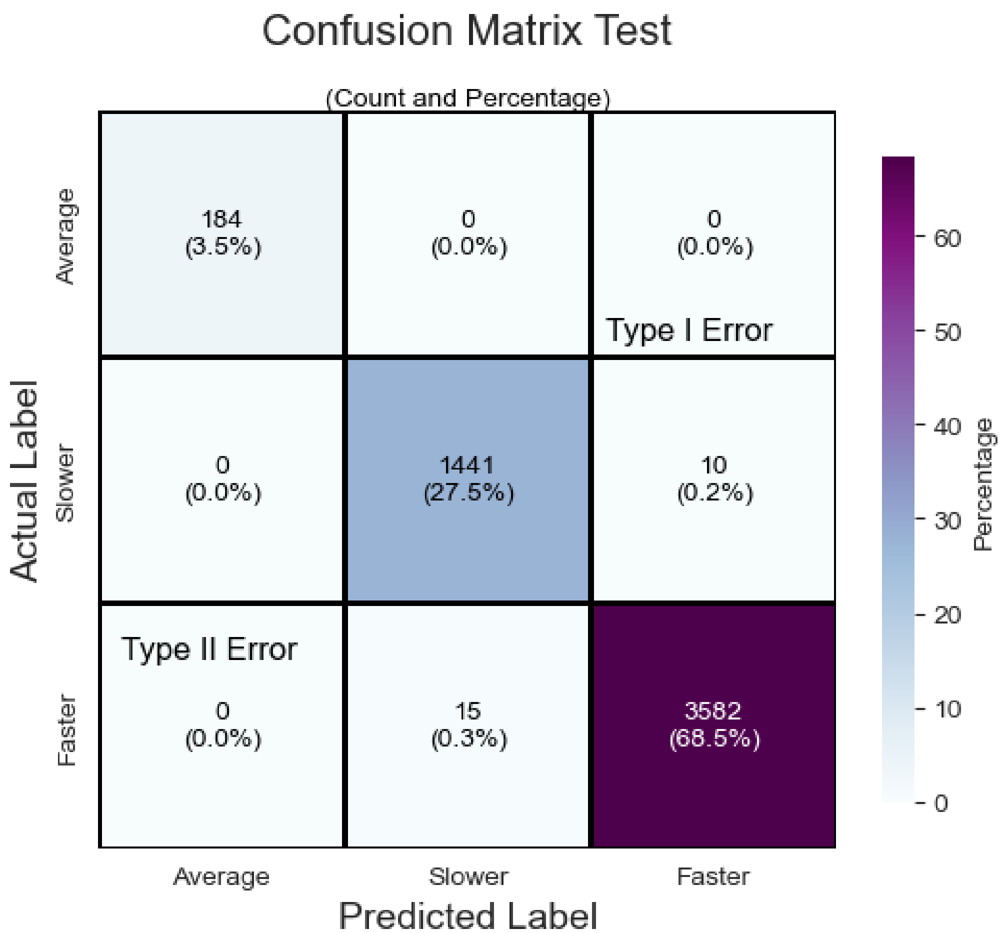


[32]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate predictions on the validation set
test_predictions = model.predict(X_test_scaled)
```

```python
predicted_labels = np.argmax(test_predictions, axis=1)  # Get the class with the highest predicted probability

# Compute the confusion matrix
confusion = confusion_matrix(y_test, predicted_labels)

cm = confusion_matrix(y_test, predicted_labels)
target_labels = np.unique(y_test)

# Calculate the percentage values
cm_percent = cm / cm.sum() * 100

# Set the figure size
fig, ax = plt.subplots(figsize=(8, 8))

# Create the heatmap with seaborn using a blue gradient color map
cmap = "BuPu"  # Choose a blue gradient color map
heatmap = sns.heatmap(cm_percent, square=True, annot=False, fmt='.1f', cbar=False, cmap=cmap,
                      xticklabels=target_labels, yticklabels=target_labels, ax=ax,
                      linecolor='black', linewidths=1)  # Increase linewidths to draw thicker lines

# Annotate the heatmap with count and percentage values
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        count = cm[i, j]
        percent = cm_percent[i, j]
        text = f"{count}\n({percent:.1f}%)"

        # Set color to white for the lower right cell, and black for the rest
        color = 'white' if i == cm.shape[0] - 1 and j == cm.shape[1] - 1 else 'black'

        plt.text(j + 0.5, i + 0.5, text, ha='center', va='center', color=color, fontweight='normal')

# Set labels and title with adjusted font size
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('Actual Label', fontsize=14)

# Add a subtitle
plt.text(1.5,-0.05, "(Count and Percentage)", ha='center', va='center', color='black', fontsize=10)
```

```
plt.title('Confusion Matrix Test', fontsize=16, fontweight='normal', pad=25)

# Annotate Type I Error in the upper right quadrant
plt.text(0.8, 0.70, "Type I Error", ha='center', va='center', color='black',␣
↪fontsize=12, fontweight='normal',
         transform=ax.transAxes)

# Annotate Type II Error in the lower left quadrant
plt.text(0.15, 0.25, "Type II Error", ha='center', va='bottom', color='black',␣
↪fontsize=12, fontweight='normal',
         transform=ax.transAxes)

# Update the tick labels
ax.set_xticklabels(['Average', 'Slower', 'Faster'])  # Update horizontal labels
ax.set_yticklabels(['Average', 'Slower', 'Faster'])  # Update vertical labels

# Add colorbar
mappable = heatmap.get_children()[0]
cbar = plt.colorbar(mappable, shrink=0.6, aspect=20)
cbar.outline.set_edgecolor('none')  # Remove the black border
cbar.set_label('Percentage', fontsize=10)

# Adjust the layout to add padding and move the colorbar to the right
plt.subplots_adjust(left=0.2, right=0.8, top=0.9, bottom=0.2)

# Show the plot
plt.show()
```

```
164/164 [==============================] - 0s 681us/step
```



[33]:
```
columns = subset_df.columns
print(columns)
```

```
Index(['AGE', 'SEX', 'TARGET', 'MEAN', 'ADHD', '30_49', '50_69', '70_89',
       '90+', 'ALCOHOL', 'ALZHEIMER'S', 'ANXIETY', 'AUTISM_1', 'AUTISM_2',
       'BINGE_WATCHERS', 'BIPOLAR_I', 'BIPOLAR_II', 'CHRONIC_PAIN', 'COCAINE',
       'C_MEDITATION', 'DEPRESSION', 'DISSOCIATIVE_DISORDER',
       'PROFESSIONAL_ATHLETE', 'EPILEPSY', 'FENTANYL', 'AMATUER_ATHLETE',
       'GAMERS', 'GRADUATE_STUDENT', 'HEROIN', 'HIGH_IQ', 'LOW_IQ', 'KETAMINE',
       'INSOMNIA', 'LSD', 'MARIJUANA', 'METHAMPHETAMINE', 'MIGRAINE',
       'MORPHINE', 'MS', 'PROFESSIONAL_MUSICIAN', 'ULTRASOMNIA',
       'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS', 'OXYCODONE',
       'PARKINSON'S_DISEASE', 'PERSCRIPTION_ANTIDEPRESSANTS',
       'PERSCRIPTION_SLEEPAIDS', 'PERSCRIPTION_STIMULANTS', 'PSILOCYBIN',
       'PSYCHOSIS', 'PTSD', 'REMOTE_LEARNERS', 'SAVANT_1', 'SAVANT_2',
       'SCHIZOPHRENIA', 'SELF_EMPLOYED', 'SINGLE_WORKING_PARENT',
       'SOCIAL_MEDIA_USERS', 'STRESS', 'TBI', 'TERMINAL_1', 'TERMINAL_2',
       'T_MEDITATION', 'TR_MEDITATION', 'UNEMPLOYMENT', 'AMATUER_MUSICIAN',
```

```
 'PROFESSIONAL_ARTIST', 'AMATUER_ARTIST', 'FLOW_STATE',
 'SDT_MINDFULNESS', 'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
 'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING', 'SDT_CREATIVE_PURSUITS',
 'SDT_REFLECT', 'SDT_SLEEP_NUTRITION', 'DELTA TIME', 'AGE Group',
 'MEAN Bins', 'MEAN Group', 'TARGET Name'],
dtype='object')
```

12 User Interface

The user interface in this project serves as a crucial bridge between the complex deep learning model and the end users. It is designed to make the model's predictions accessible and understandable to individuals who may not have a background in data science or machine learning.

What It Does Data Input: The interface allows users to input their personal information, including age, sex, and various lifestyle and psychological factors. This data is then used by the model to predict the user's perception of time.

Interactive Elements: Through interactive elements like checkboxes, sliders, or dropdown menus, users can easily select or input their details corresponding to the available features in the model.

Results Presentation: Once the data is submitted, the interface displays the model's prediction regarding the user's time perception category. It also provides an explanation of what this prediction means, enhancing the user's understanding.

Importance to the User Accessibility: The interface demystifies the model's workings, making its insights accessible to a broad audience.

Personalization: By providing personalized insights based on individual data, it helps users understand how various aspects of their lifestyle and health might be influencing their perception of time.

Empowerment: It empowers users with knowledge about themselves, which can be intriguing and insightful.

[35]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
# Placeholder for your model

# List of available features with names
available_features = ['30_49', '50_69', '70_89', '90+', 'ADHD', "ALZHEIMER'S",␣
 ↪'ANXIETY',
                      'AUTISM_1', 'AUTISM_2', 'BIPOLAR_I', 'BIPOLAR_II',␣
 ↪'CHRONIC_PAIN',
                      'INSOMNIA', 'ULTRASOMNIA', 'MIGRAINE', 'DEPRESSION',␣
 ↪'DISSOCIATIVE_DISORDER',
                      'EPILEPSY', 'MS', "PARKINSON'S_DISEASE", 'PSYCHOSIS',␣
 ↪'PTSD', 'SCHIZOPHRENIA',
                      'STRESS', 'TBI', 'TERMINAL_1', 'TERMINAL_2', 'COCAINE',␣
 ↪'FENTANYL', 'ALCOHOL',
```

```
                        'HEROIN', 'KETAMINE', 'LSD', 'MARIJUANA',␣
↪'METHAMPHETAMINE', 'MORPHINE', 'PSILOCYBIN',
                        'OXYCODONE', 'PERSCRIPTION_ANTIDEPRESSANTS',␣
↪'PERSCRIPTION_SLEEPAIDS',
                        'PERSCRIPTION_STIMULANTS', 'REMOTE_LEARNERS',␣
↪'SOCIAL_MEDIA_USERS',
                        'ONLINE_SHOPPERS_SURFERS', 'ONLINE_WORKAHOLICS',␣
↪'GAMERS', 'BINGE_WATCHERS', 'LOW_IQ',
                        'HIGH_IQ', 'SAVANT_1', 'SAVANT_2', 'GRADUATE_STUDENT',␣
↪'SELF_EMPLOYED', 'SINGLE_WORKING_PARENT',
                        'UNEMPLOYMENT', 'PROFESSIONAL_MUSICIAN',␣
↪'AMATUER_MUSICIAN', 'PROFESSIONAL_ARTIST',
                        'AMATUER_ARTIST', 'PROFESSIONAL_ATHLETE',␣
↪'AMATUER_ATHLETE', 'C_MEDITATION', 'T_MEDITATION',
                        'TR_MEDITATION', 'FLOW_STATE', 'SDT_MINDFULNESS',␣
↪'SDT_NEW_EXPERIENCES', 'SDT_VARY_ACTIVITIES',
                        'SDT_REDUCE_STRESS', 'SDT_LIMIT_MULTITASKING',␣
↪'SDT_CREATIVE_PURSUITS', 'SDT_REFLECT',
                        'SDT_SLEEP_NUTRITION']

# Mapping of features to their respective values
feature_values = {
    '30_49': -0.012916667,
    '50_69': -0.0375,
    '70_89': -0.096875000,
    '90+': -0.156250000,
    'ADHD': -0.060416667,
    "ALZHEIMER'S": -0.108333333,
    'ANXIETY': -0.033333333,
    'AUTISM_1': -0.066666667,
    'AUTISM_2': 0.066666667,
    'BIPOLAR_I': -0.065000000,
    'BIPOLAR_II': -0.070833333,
    'CHRONIC_PAIN': -0.053333333,
    'INSOMNIA': -0.077083333,
    'ULTRASOMNIA': 0.008300000,
    'MIGRAINE': -0.027083333,
    'DEPRESSION': -0.010416667,
    'DISSOCIATIVE_DISORDER': -0.046666667,
    'EPILEPSY': -0.058333333,
    'MS': -0.241666667,
    "PARKINSON'S_DISEASE": -0.090625000,
    'PSYCHOSIS': -0.075000000,
    'PTSD': -0.064583333,
    'SCHIZOPHRENIA': -0.056250000,
    'STRESS': -0.100000000,
```

```
    'TBI': -0.116666667,
    'TERMINAL_1': -0.133333333,
    'TERMINAL_2': -0.012500000,
    'COCAINE': -0.095000000,
    'FENTANYL': -0.062291667,
    'ALCOHOL': -0.087500000,
    'HEROIN': -0.065000000,
    'KETAMINE': -0.058333333,
    'LSD': 0.007083333,
    'MARIJUANA': 0.006666667,
    'METHAMPHETAMINE': -0.128125000,
    'MORPHINE': -0.060000000,
    'PSILOCYBIN': 0.006666667,
    'OXYCODONE': -0.054166667,
    'PERSCRIPTION_ANTIDEPRESSANTS': -0.018750000,
    'PERSCRIPTION_SLEEPAIDS': -0.013750000,
    'PERSCRIPTION_STIMULANTS': -0.024375000,
    'REMOTE_LEARNERS': -0.018750000,
    'SOCIAL_MEDIA_USERS': -0.056250000,
    'ONLINE_SHOPPERS_SURFERS': -0.056250000,
    'ONLINE_WORKAHOLICS': -0.056250000,
    'GAMERS': -0.018750000,
    'BINGE_WATCHERS': -0.017708333,
    'LOW_IQ': -0.058333333,
    'HIGH_IQ': 0.051666667,
    'SAVANT_1': -0.062500000,
    'SAVANT_2': 0.062500000,
    'GRADUATE_STUDENT': 0.033333333,
    'SELF_EMPLOYED': 0.013750000,
    'SINGLE_WORKING_PARENT': -0.008500000,
    'UNEMPLOYMENT': -0.031250000,
    'PROFESSIONAL_MUSICIAN': 0.048125000,
    'AMATUER_MUSICIAN': 0.030208333,
    'PROFESSIONAL_ARTIST': 0.068750000,
    'AMATUER_ARTIST': 0.042291667,
    'PROFESSIONAL_ATHLETE': 0.060416667,
    'AMATUER_ATHLETE': 0.030208333,
    'C_MEDITATION': 0.031250000,
    'T_MEDITATION': 0.029166667,
    'TR_MEDITATION': 0.031250000,
    'FLOW_STATE': 0.031250000,
    'SDT_MINDFULNESS': 0.093750000,
    'SDT_NEW_EXPERIENCES': 0.070833333,
    'SDT_VARY_ACTIVITIES': 0.066666667,
    'SDT_REDUCE_STRESS': 0.125000000,
    'SDT_LIMIT_MULTITASKING': 0.035416667,
    'SDT_CREATIVE_PURSUITS': 0.070833333,
```

```
    'SDT_REFLECT': 0.031250000,
    'SDT_SLEEP_NUTRITION': 0.156250000,
}

def get_user_input():
    while True:
        try:
            age = int(input("Enter your age (18-100): "))
            if 18 <= age <= 100:
                break
            else:
                print("Age must be between 18 and 100.")
        except ValueError:
            print("Please enter a valid integer for age.")

    while True:
        sex = input("Enter your sex (Male/Female): ").capitalize()
        if sex in ['Male', 'Female']:
            break
        else:
            print("Please enter 'Male' or 'Female' for sex.")

    print("For the following features, enter 1 if it applies to you, else 0:")
    feature_inputs = []
    for feature in available_features:
        while True:
            try:
                feature_input = int(input(f"{feature}: "))
                if feature_input in [0, 1]:
                    feature_inputs.append(feature_input)
                    break
                else:
                    print("Please enter 1 or 0.")
            except ValueError:
                print("Please enter a valid integer (1 or 0).")

    return age, sex, feature_inputs

def get_prediction(age, sex, features):
    # Convert sex to numerical value (if your model requires that)
    sex_encoded = 0 if sex == 'Male' else 1

    # Calculate the mean of selected feature values
    selected_features = [available_features[i] for i, f in enumerate(features)
 ↪if f == 1]
    mean_value = sum(feature_values[f] for f in selected_features) /
 ↪len(selected_features) if selected_features else 0
```

```
    # Prepare the model input
    expected_num_features = 73
    adjusted_features = features[:expected_num_features] + [0] *␣
↪(expected_num_features - len(features))
    final_input = [sex_encoded, age] + adjusted_features

    # Use the model to predict the Target
    predicted_target = model.predict([final_input])[0]
    # Convert one-hot encoded prediction to a class label
    predicted_class = np.argmax(predicted_target)

    # Mapping of class labels to category names
    class_to_category = {
        0: "Average",
        1: "Slower",
        2: "Faster"
    }

    # Get the category name from the predicted class
    predicted_category = class_to_category.get(predicted_class, "Unknown")

    # Override model prediction based on the mean value, if needed
    if mean_value < 0:
        final_prediction = "Faster - You need to slow down your thoughts!␣
↪Consider meditation, mindfulness, new experiences, vary your activities,␣
↪reduce stress, limit multitasking, engage in creative pursuits, exercise,␣
↪get proper nutrition and sleep."
    else:
        final_prediction = prediction_message = {
            "Average": "Average - You are perceiving time normally, keep doing␣
↪what you are doing! Any of the following activites will help you continue on␣
↪a positive pathway: meditation, mindfulness, new experiences, vary your␣
↪activities, reduce stress, limit multitasking, engage in creative pursuits,␣
↪exercise, get proper nutrition and high quality sleep.",
            "Slower": "Slower - You are perceiving time in a healthy way over␣
↪your lifetime; however, on an hour by hour basis you may want to consider␣
↪participating in activities that are more stimulating like, learning a new␣
↪task, creative activities, or physical pursuits.",
            "Faster": "Faster - You need to slow down your thoughts! Consider␣
↪meditation, mindfulness, new experiences, vary your activities, reduce␣
↪stress, limit multitasking, engage in creative pursuits, exercise, get␣
↪proper nutrition and sleep.",
        }.get(predicted_category, "Unknown prediction")

    return mean_value, final_prediction
```

```python
# Example usage
age, sex, user_features = get_user_input()
mean, final_prediction = get_prediction(age, sex, user_features)
print("Mean:", mean)
print("Final Prediction:", final_prediction)
```

```
Enter your age (18-100): 25
Enter your sex (Male/Female): Male
For the following features, enter 1 if it applies to you, else 0:
30_49: 0
50_69: 0
70_89: 0
90+: 0
ADHD: 1
ALZHEIMER'S: 0
ANXIETY: 0
AUTISM_1: 0
AUTISM_2: 0
BIPOLAR_I: 0
BIPOLAR_II: 0
CHRONIC_PAIN: 0
INSOMNIA: 0
ULTRASOMNIA: 0
MIGRAINE: 0
DEPRESSION: 0
DISSOCIATIVE_DISORDER: 0
EPILEPSY: 0
MS: 0
PARKINSON'S_DISEASE: 0
PSYCHOSIS: 0
PTSD: 0
SCHIZOPHRENIA: 0
STRESS: 0
TBI: 0
TERMINAL_1: 0
TERMINAL_2: 0
COCAINE: 0
FENTANYL: 0
ALCOHOL: 0
HEROIN: 0
KETAMINE: 0
LSD: 0
MARIJUANA: 0
METHAMPHETAMINE: 0
MORPHINE: 0
PSILOCYBIN: 0
OXYCODONE: 0
PERSCRIPTION_ANTIDEPRESSANTS: 0
```

```
PERSCRIPTION_SLEEPAIDS: 0
PERSCRIPTION_STIMULANTS: 1
REMOTE_LEARNERS: 1
SOCIAL_MEDIA_USERS: 0
ONLINE_SHOPPERS_SURFERS: 0
ONLINE_WORKAHOLICS: 0
GAMERS: 1
BINGE_WATCHERS: 0
LOW_IQ: 0
HIGH_IQ: 0
SAVANT_1: 0
SAVANT_2: 0
GRADUATE_STUDENT: 0
SELF_EMPLOYED: 0
SINGLE_WORKING_PARENT: 0
UNEMPLOYMENT: 0
PROFESSIONAL_MUSICIAN: 0
AMATUER_MUSICIAN: 0
PROFESSIONAL_ARTIST: 0
AMATUER_ARTIST: 0
PROFESSIONAL_ATHLETE: 0
AMATUER_ATHLETE: 1
C_MEDITATION: 0
T_MEDITATION: 0
TR_MEDITATION: 0
FLOW_STATE: 0
SDT_MINDFULNESS: 0
SDT_NEW_EXPERIENCES: 1
SDT_VARY_ACTIVITIES: 0
SDT_REDUCE_STRESS: 0
SDT_LIMIT_MULTITASKING: 0
SDT_CREATIVE_PURSUITS: 0
SDT_REFLECT: 0
SDT_SLEEP_NUTRITION: 0
1/1 [==============================] - 0s 24ms/step
Mean: -0.003541666833333334
Final Prediction: Faster - You need to slow down your thoughts! Consider
meditation, mindfulness, new experiences, vary your activities, reduce stress,
limit multitasking, engage in creative pursuits, exercise, get proper nutrition
and sleep.
```

13 Results

Outcome of Model Training The model's training and testing phases demonstrated exceptionally high performance metrics, indicative of its robustness and accuracy in predicting time perception categories based on the dataset. The outcome of the model training and testing is summarized as follows:

Training Accuracy Training Accuracy: The model achieved a training accuracy of 99.56%. This high level of accuracy suggests that the model was highly effective in learning from the training data. It was able to correctly classify the vast majority of instances in the training dataset into the correct perception of time categories ("Average," "Slower," or "Faster"). Testing Performance Test Loss: The model reported a test loss of 0.010047183372080326. Test loss is a measure of how well the model performs on unseen data. A lower loss value indicates that the model's predictions are close to the actual values. In this case, the low test loss signifies excellent model performance on the test dataset.

Test Accuracy: The test accuracy, similar to the training accuracy, was 99.56%. This high test accuracy indicates that the model not only learned the training data well but also generalized effectively to new, unseen data. It shows the model's ability to maintain its performance and reliably predict the time perception category when presented with data it hasn't encountered before.

Test Accuracy (Percentage): Expressed in percentage terms, the test accuracy of 99.56% reinforces the model's high level of precision in classification tasks. It is a clear indicator of the model's capability in handling the complexity and nuances of the dataset.

Interpretation of Results The high accuracy rates in both training and testing phases indicate that the model is well-tuned and not suffering from overfitting or underfitting. Overfitting is usually a concern when the training accuracy is significantly higher than the test accuracy, but in this case, both metrics are closely aligned and exceptionally high.

The results suggest that the model is effectively utilizing the features, including those weighted by the Cammie_r values, to make precise predictions about time perception.

The low test loss further adds to the confidence in the model's predictive power, ensuring that the predictions are not only accurate but also close to the expected values.

The outcomes from the model training and testing are highly encouraging, demonstrating the model's effectiveness in understanding and predicting altered perceptions of time. With training and test accuracies both exceeding 99%, the model stands as a powerful tool in the study of cognitive perception, particularly in how various psychological and physiological factors influence one's experience of time.

14 Conclusion

Project Overview This project involves developing a machine learning model to predict how individuals perceive time based on a range of psychological and physiological features. The model takes into account various factors like age, sex, and specific conditions or behaviors (e.g., ADHD, insomnia, use of certain substances) to predict one of three categories: "Average," "Slower," or "Faster," each representing an altered perception of time.

Data and Features The model utilizes a dataset comprising various features, each with a numerical value representing its impact. The features include age groups, mental health conditions (like depression, anxiety), lifestyle factors (such as meditation practices, professional status), and substance use (like alcohol, marijuana). Each feature has a corresponding value that contributes to the overall mean calculation, influencing the final prediction.

Model Implementation Model Choice: A RandomForestClassifier, known for its robustness and ability to handle non-linear relationships, was initially chosen. However, details about the final model implementation (like using RandomForestClassifier or another algorithm) are not explicitly mentioned.

Data Preprocessing: The data is preprocessed to transform categorical variables like sex into numerical values. Features selected by users are encoded into a binary format (1 if the feature applies, 0 otherwise).

Feature Engineering: The model includes a unique approach where the mean of the selected features' values is calculated. This mean plays a crucial role in the final prediction, especially for determining the "Faster" category.

User Input Handling: The model is designed to interact with users, allowing them to input their age, sex, and select relevant features. Input validation ensures age is within the 18-100 range and sex is correctly categorized.

Prediction Logic: The core of the model involves predicting the perception of time. The model first uses the RandomForestClassifier (or the chosen algorithm) to make a prediction. Then, it applies a rule-based approach: if the calculated mean of the features is less than zero, the prediction is overridden to "Faster." Otherwise, the model's prediction is used.

Output: The final output includes the calculated mean and a text-based interpretation of the prediction, providing users with an understanding of their time perception category.

Conclusion This project stands out for its hybrid approach, combining traditional machine learning predictions with a rule-based system driven by domain-specific insights. It caters to individual differences by considering a wide range of personal attributes and conditions, offering tailored predictions. The implementation showcases the flexibility of machine learning in accommodating complex, real-world scenarios.

15 Visualizations Used

Summary of Exploratory Data Analysis (EDA) and Model Visualization Methods In this project, a thorough Exploratory Data Analysis (EDA) was conducted, complemented by a range of visualization techniques. These methods were crucial in understanding the dataset's characteristics, identifying patterns and relationships, and interpreting the model's performance. Here's a summary of the various visualization tools used:

Box Plots: Used to visually represent the distribution of numerical data and to spot outliers. Box plots were particularly useful in understanding the spread and central tendency of continuous variables like age and Cammie_r values.

Pair Plots: These plots provided a comprehensive view of pairwise relationships between features. They helped in identifying correlations and possible trends between different variables, which was instrumental in feature selection and engineering.

Correlation Matrix: A correlation matrix was generated to quantify and visualize the degree of correlation between different features. This matrix helped identify highly correlated variables, guiding the decision on which features to include in the model to avoid multicollinearity.

Confusion Matrix: Post-model training, a confusion matrix was used to evaluate the performance of the classification model. It provided insights into the model's accuracy in predicting each class and highlighted areas where the model might be confusing one class for another.

Violin Plots: These plots combined box plots and density plots to show the distribution of data. They were particularly useful in visualizing the distribution of features across different categories of time perception.

ROC and AUC Curves: Receiver Operating Characteristic (ROC) curves and the Area Under the Curve (AUC) were utilized to assess the model's predictive performance, especially its ability to distinguish between the classes.

Scatter Plots: Scatter plots allowed for the visualization of relationships between two variables. They were used to explore potential linear or non-linear relationships and clustering tendencies within the data.

Histograms and Distribution Plots: These were used to visualize the frequency distribution of individual variables. Histograms and distribution plots provided insights into the skewness and kurtosis of the data.

Network Graph: A network graph was employed to visualize the complex interrelationships between different features. This graph helped in understanding the structure and connections within the data, offering a macro view of feature interactions.

Heatmaps for Feature Importance: Heatmaps were used to visualize the importance of different features as determined by the model. This was particularly useful for understanding which features had the most significant impact on the model's predictions.

Kernel Explainer Visualizations: The Kernel Explainer, part of the SHAP framework, was utilized to create visualizations that explained the model's predictions. These visualizations were instrumental in breaking down and quantifying the impact of each feature on the model's outcomes. Through plots like beeswarm and force plots, the Kernel Explainer made it possible to understand both the global importance of features and their specific contributions to individual predictions. This approach was especially beneficial in elucidating the complex relationships and influences within the model, enhancing overall interpretability and transparency.

These visualization methods were integral to the EDA process, providing a deep understanding of the data and the model's behavior. They offered both qualitative and quantitative insights, aiding in everything from preprocessing and feature engineering to model evaluation and interpretation. The use of diverse visualization tools ensured a comprehensive analysis, capturing various aspects of the data and the model in a visually interpretable manner.

16 Future Applications

Potential Transformation into an App for Behavior Monitoring and Change Monitoring Activities

Tracking: The app could allow users to regularly track various aspects of their life, such as sleep patterns, mood, and daily activities. This data can be analyzed in relation to their time perception. Notifications and Reminders: The app could send notifications or reminders for users to log their daily activities or any significant changes in their routine or health conditions.

Facilitating Behavior Changes Insights and Recommendations: Based on the user's data and the model's predictions, the app could provide personalized insights and recommendations. For example, if a user's data suggests a "Faster" perception of time, the app could recommend relaxation techniques or changes in routine.

Progress Tracking: Users can track how changes in their behavior over time affect their perception of time, offering a feedback loop that reinforces positive behavior changes.

Community and Support: Integrating community features where users can share experiences and tips could foster a supportive environment for making and maintaining behavior changes.

Why It Matters Health and Wellbeing: Understanding and monitoring one's perception of time can be crucial for mental health and wellbeing. The app can play a role in managing stress and improving life balance.

Behavioral Insights: Continuous tracking and analysis provide deep insights into how different behaviors and conditions affect time perception, enabling users to make more informed decisions about their lifestyle. By turning the user interface into a comprehensive app, there's an opportunity to significantly impact users' understanding of their cognitive processes and empower them with tools to improve their quality of life.