```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Cleaned data from the wellbeing and lifestyle kaggle dataset
data = pd.read_csv("data_clean.csv") # Unscaled
# Cleaned training, validation, and test data from same dataset
train_data = pd.read_csv("data_train.csv")
val_data = pd.read_csv("data_val.csv")
test_data = pd.read_csv("data_test.csv")

train_data.head()
```

```
   FRUITS_VEGGIES  DAILY_STRESS  PLACES_VISITED  CORE_CIRCLE  \
0       -0.636358      0.867892       -0.376228     0.519366
1       -1.328086      0.139683        1.431552     1.569497
2       -0.636358      0.867892       -0.376228    -0.530764
3        0.747098     -1.316734       -0.677524     0.869410
4        0.747098     -1.316734        1.431552     1.569497

   SUPPORTING_OTHERS  SOCIAL_NETWORK  ACHIEVEMENT  DONATION  BMI_RANGE  \
0           0.416268        1.138570     0.002509  0.688858  -0.836204
1          -0.815281       -0.157744    -0.726154  1.230072  -0.836204
2          -0.507394       -0.157744     0.731173  0.688858   1.195880
3           0.724156        1.138570     1.824168  1.230072  -0.836204
4           1.339930        0.490413     1.095505  1.230072  -0.836204

   TODO_COMPLETED  ...  PERSONAL_AWARDS  TIME_FOR_PASSION  WEEKLY_MEDITATION  \
0        0.097801  ...        -0.231363         -0.476632          -0.738927
1        0.097801  ...        -0.231363         -1.210773           0.253374
2        0.097801  ...         1.392280         -0.843703          -0.738927
3       -1.042719  ...         1.392280          0.624579           1.245675
4        0.477974  ...        -0.880820          0.991650           0.253374

   WORK_LIFE_BALANCE_SCORE  AGE_21 to 35  AGE_36 to 50  AGE_51 or more  \
0                 0.463390           0.0           0.0             0.0
1                 0.117818           0.0           1.0             0.0
2                -0.455164           0.0           0.0             1.0
3                 1.121093           0.0           0.0             1.0
4                 2.115450           0.0           0.0             1.0

   AGE_Less than 20  GENDER_Female  GENDER_Male
0               1.0            1.0          0.0
1               0.0            1.0          0.0
2               0.0            1.0          0.0
3               0.0            1.0          0.0
4               0.0            1.0          0.0
```

```
[5 rows x 27 columns]
```

```
val_data.head()
```

```
   FRUITS_VEGGIES  DAILY_STRESS  PLACES_VISITED  CORE_CIRCLE  \
0        0.055370      0.139683       -1.581414    -1.230851
1        0.747098     -0.588525       -0.677524     1.219453
2        0.055370      0.139683        0.226365    -0.180721
3        0.055370     -1.316734       -0.074931     0.519366
4        0.055370      0.867892       -1.280117     1.569497

   SUPPORTING_OTHERS  SOCIAL_NETWORK  ACHIEVEMENT  DONATION  BMI_RANGE  \
0          -1.123169       -1.454058    -1.454817 -1.475996  -0.836204
1           1.339930        1.138570     1.459836 -0.393569  -0.836204
2           1.339930        1.138570     0.731173  1.230072  -0.836204
3          -0.507394       -1.129979    -0.726154 -1.475996  -0.836204
4           1.032043        0.490413     0.366841 -0.934782   1.195880

   TODO_COMPLETED  ...  PERSONAL_AWARDS  TIME_FOR_PASSION  WEEKLY_MEDITATION  \
0        0.858147  ...        -1.205549         -1.210773           0.253374
1        0.858147  ...         0.418094          0.257509           1.245675
2       -0.282372  ...        -0.880820          0.991650           1.245675
3        0.097801  ...        -0.880820         -1.210773           1.245675
4        0.477974  ...        -0.880820         -0.843703           1.245675

   WORK_LIFE_BALANCE_SCORE  AGE_21 to 35  AGE_36 to 50  AGE_51 or more  \
0                -1.048211           0.0           0.0             1.0
1                 1.103257           0.0           0.0             1.0
2                 1.315060           0.0           1.0             0.0
3                 0.371981           0.0           0.0             0.0
4                -0.660278           0.0           1.0             0.0

   AGE_Less than 20  GENDER_Female  GENDER_Male
0               0.0            1.0          0.0
1               0.0            0.0          1.0
2               0.0            0.0          1.0
3               1.0            0.0          1.0
4               0.0            1.0          0.0

[5 rows x 27 columns]
```

```
test_data.head()
```

```
   FRUITS_VEGGIES  DAILY_STRESS  PLACES_VISITED  CORE_CIRCLE  \
0       -0.636358      0.139683       -0.376228    -0.880808
1        0.055370     -0.588525       -0.074931     0.169323
2       -0.636358      0.139683       -1.581414    -1.230851
```

```
3      -1.328086      0.867892       -0.677524    -1.580895
4      -1.328086      1.596101       -0.978821    -1.580895

   SUPPORTING_OTHERS  SOCIAL_NETWORK  ACHIEVEMENT  DONATION  BMI_RANGE  \
0          -1.431056       -0.805901    -1.090486 -1.475996  -0.836204
1          -1.738944        1.138570    -0.361822 -0.393569  -0.836204
2           1.339930       -1.454058     1.095505  1.230072   1.195880
3          -0.815281       -0.481822    -0.361822 -0.934782  -0.836204
4          -1.738944       -1.778136    -1.090486 -1.475996  -0.836204

   TODO_COMPLETED  ...  PERSONAL_AWARDS  TIME_FOR_PASSION  WEEKLY_MEDITATION  \
0        0.097801  ...        -0.880820         -0.109562          -0.408160
1       -1.422892  ...        -0.231363         -0.843703           0.584141
2        0.477974  ...         0.093365         -0.476632          -1.400461
3       -1.422892  ...        -0.880820         -0.843703          -1.400461
4       -0.662546  ...        -0.880820         -1.210773          -1.731228

   WORK_LIFE_BALANCE_SCORE  AGE_21 to 35  AGE_36 to 50  AGE_51 or more  \
0                -1.019228           0.0           0.0             0.0
1                -0.716016           1.0           0.0             0.0
2                -0.247821           0.0           0.0             1.0
3                -1.625652           1.0           0.0             0.0
4                -2.468403           1.0           0.0             0.0

   AGE_Less than 20  GENDER_Female  GENDER_Male
0               1.0            0.0          1.0
1               0.0            1.0          0.0
2               0.0            0.0          1.0
3               0.0            1.0          0.0
4               0.0            1.0          0.0

[5 rows x 27 columns]
```

Task 2: Build and Train Various ML Models

```python
# Part 1: Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Prepare the datasets
X_train = train_data.drop(['GENDER_Female', 'GENDER_Male'], axis=1)
y_train = train_data['GENDER_Male']
X_val = val_data.drop(['GENDER_Female', 'GENDER_Male'], axis=1)
y_val = val_data['GENDER_Male']
X_test = test_data.drop(['GENDER_Female', 'GENDER_Male'], axis=1)
```

```python
y_test = test_data['GENDER_Male']

# Initialize and train the logistic regression model
max_iter_val = 5000
# Adjusting the hyperparameters to what gave the highest found accuracy (based on cell belo
log_reg = LogisticRegression(max_iter=max_iter_val)
log_reg.fit(X_train, y_train)

y_train_pred = log_reg.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy}")

# Predict on the validation set
y_val_pred = log_reg.predict(X_val)

# Evaluate the model on the validation set
val_accuracy = accuracy_score(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)

print(f"Validation Accuracy: {val_accuracy}")
print("Confusion Matrix:\n", val_conf_matrix)
```

```
Training Accuracy: 0.6669797537048633
Validation Accuracy: 0.6651017214397497
Confusion Matrix:
 [[1701  264]
 [ 806  424]]
```

```python
# Trying out different hyperparameters
C_values = [0.001, 0.01, 0.1, 1, 10, 100]
penalties = ['l1', 'l2', 'elasticnet', 'none']
```

```python
from sklearn.metrics import accuracy_score

best_accuracy = 0
best_params = {'C': None, 'penalty': None}

for penalty in penalties:
    # Skip invalid solver/penalty combinations
    if penalty == 'elasticnet':
        solver = 'saga'
    else:
        solver = 'liblinear'

    for C in C_values:
        # Initialize the Logistic Regression model with current parameters
        if penalty == 'elasticnet':
```

```
            model = LogisticRegression(C=C, penalty=penalty, l1_ratio=0.5, solver=solver, ma
        else:
            model = LogisticRegression(C=C, penalty=penalty, solver=solver, max_iter=1000)

        # Fitting the model on training data and predicting on validation data
        try:
            model.fit(X_train, y_train)
            y_val_pred = model.predict(X_val)
            accuracy = accuracy_score(y_val, y_val_pred)

            # printing performance and update the best parameters
            print(f"Penalty: {penalty}, C: {C}, Accuracy: {accuracy}")
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_params['C'] = C
                best_params['penalty'] = penalty
        except Exception as e:
            # This block catches errors which might arise from invalid parameter combinatio
            print(f"Could not train model with Penalty: {penalty}, C: {C}. Error: {str(e)}")

best_model = LogisticRegression(C=best_params['C'], penalty=best_params['penalty'], solver='
best_model.fit(X_train, y_train)
y_train_pred = best_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"SVM Training Accuracy: {train_accuracy}")
# Print out the best parameters and best accuracy found
print(f"Best parameters: {best_params}")
print(f"Best validation accuracy: {best_accuracy}")
```

```
Penalty: l1, C: 0.001, Accuracy: 0.6150234741784038
Penalty: l1, C: 0.01, Accuracy: 0.6613458528951487
Penalty: l1, C: 0.1, Accuracy: 0.6613458528951487
Penalty: l1, C: 1, Accuracy: 0.6644757433489827
Penalty: l1, C: 10, Accuracy: 0.6651017214397497
Penalty: l1, C: 100, Accuracy: 0.6651017214397497
Penalty: l2, C: 0.001, Accuracy: 0.6629107981220658
Penalty: l2, C: 0.01, Accuracy: 0.6632237871674491
Penalty: l2, C: 0.1, Accuracy: 0.6644757433489827
Penalty: l2, C: 1, Accuracy: 0.6651017214397497
Penalty: l2, C: 10, Accuracy: 0.6651017214397497
Penalty: l2, C: 100, Accuracy: 0.6651017214397497
Penalty: elasticnet, C: 0.001, Accuracy: 0.6150234741784038
Penalty: elasticnet, C: 0.01, Accuracy: 0.662284820031299
Penalty: elasticnet, C: 0.1, Accuracy: 0.6632237871674491
Penalty: elasticnet, C: 1, Accuracy: 0.6647887323943662
Penalty: elasticnet, C: 10, Accuracy: 0.6651017214397497
```

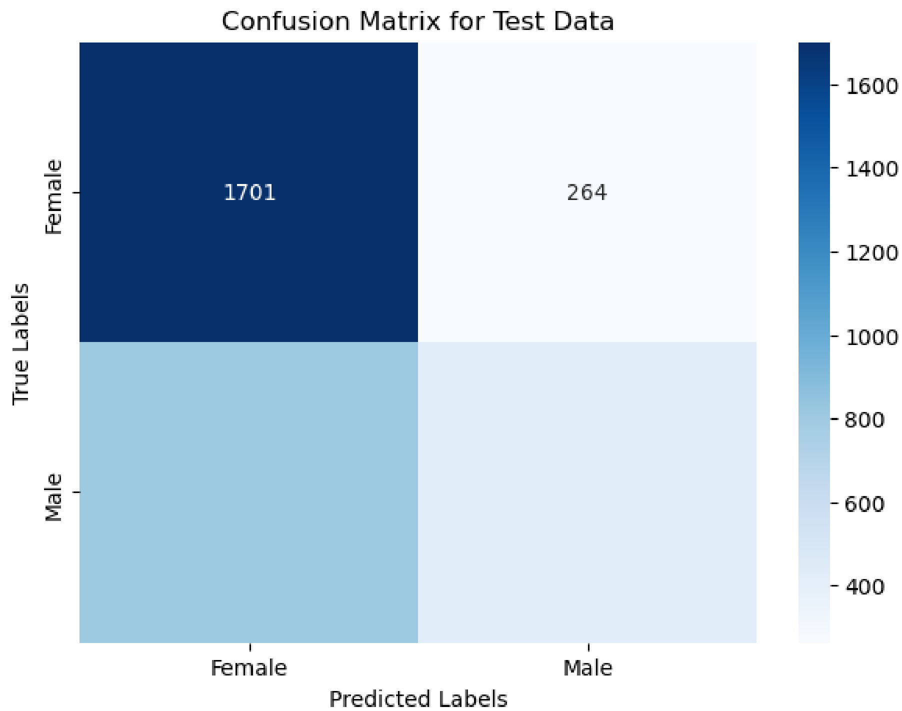
```
/Users/madmax11/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350: Co
  warnings.warn(
```

```
Penalty: elasticnet, C: 100, Accuracy: 0.6651017214397497
Could not train model with Penalty: none, C: 0.001. Error: penalty='none' is not supported f
Could not train model with Penalty: none, C: 0.01. Error: penalty='none' is not supported fo
Could not train model with Penalty: none, C: 0.1. Error: penalty='none' is not supported for
Could not train model with Penalty: none, C: 1. Error: penalty='none' is not supported for t
Could not train model with Penalty: none, C: 10. Error: penalty='none' is not supported for
Could not train model with Penalty: none, C: 100. Error: penalty='none' is not supported for
SVM Training Accuracy: 0.6669797537048633
Best parameters: {'C': 10, 'penalty': 'l1'}
Best validation accuracy: 0.6651017214397497
```

```
#Optimized model with new beset found hyper parameters
log_reg_optimized = LogisticRegression(max_iter=5000, C=0.01, penalty='elasticnet', solver='
log_reg_optimized.fit(X_train, y_train)

#Prediction on test set
y_test_pred = log_reg_optimized.predict(X_test)

# Evaluate the model on the validation set
test_accuracy = accuracy_score(y_val, y_val_pred)
test_conf_matrix = confusion_matrix(y_val, y_val_pred)

print(f"Test Accuracy: {test_accuracy}")
print("Confusion Matrix:\n", test_conf_matrix)

plt.figure(figsize=(7, 5))
sns.heatmap(val_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Female', 'Male
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Test Data')
```

```
Test Accuracy: 0.6651017214397497
Confusion Matrix:
 [[1701  264]
 [ 806  424]]
```

```
Text(0.5, 1.0, 'Confusion Matrix for Test Data')
```



1. Support Vector Machine

```
from sklearn.svm import SVC

# Training model and checking different hyperparameters
kernels = ['linear', 'rbf', 'poly']
for kernel in kernels:
    svm_model = SVC(kernel=kernel, C=1.0)
    if kernel == 'poly':
        svm_model.degree = 3
    svm_model.fit(X_train, y_train)
    y_val_pred = svm_model.predict(X_val)
    val_accuracy = accuracy_score(y_val, y_val_pred)
    print(f'Validation Accuracy with {kernel} kernel: {val_accuracy}')
```

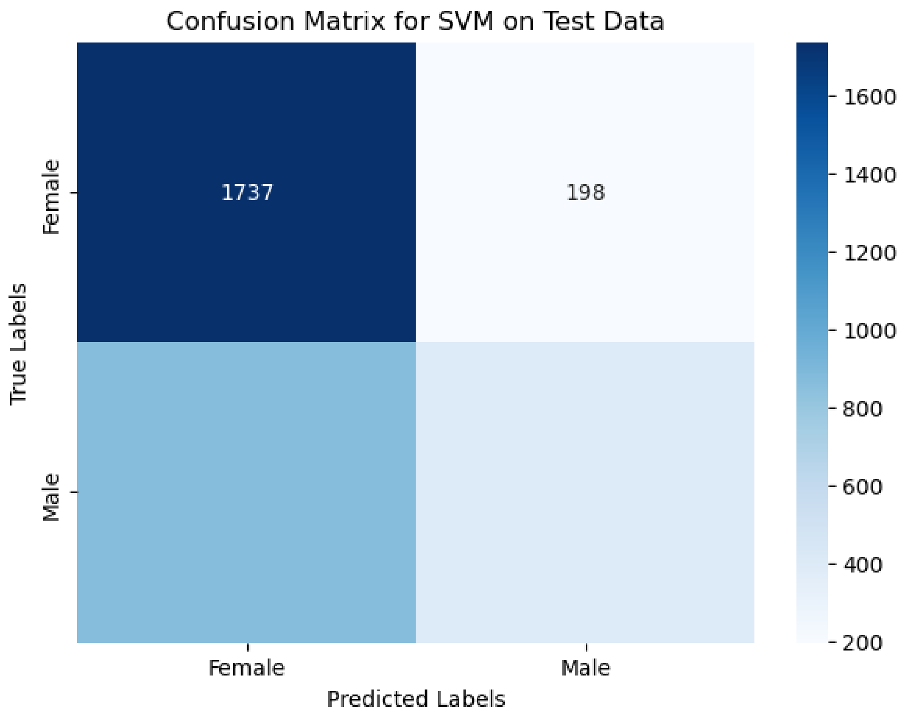
```
Validation Accuracy with linear kernel: 0.6528951486697966
Validation Accuracy with rbf kernel: 0.6666666666666666
Validation Accuracy with poly kernel: 0.6519561815336463
```

```
#Optimized model
svm_optimized_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_optimized_model.fit(X_train, y_train)

y_test_pred = svm_optimized_model.predict(X_test)
```

```
test_accuracy = accuracy_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

print(f"Test Accuracy: {test_accuracy}")
print("Confusion Matrix for test data:\n", test_conf_matrix)

plt.figure(figsize=(7, 5))
sns.heatmap(test_conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Female', 'Mal
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for SVM on Test Data')
```

```
Test Accuracy: 0.6685446009389672
Confusion Matrix for test data:
 [[1737  198]
 [ 861  399]]
```

```
Text(0.5, 1.0, 'Confusion Matrix for SVM on Test Data')
```



1. Decision Trees

```
age_columns = ['AGE_21 to 35', 'AGE_36 to 50', 'AGE_51 or more', 'AGE_Less than 20']

for df in [train_data, val_data, test_data]:
```

```
    conditions = [
        df['AGE_21 to 35'] == 1,
        df['AGE_36 to 50'] == 1,
        df['AGE_51 or more'] == 1,
        df['AGE_Less than 20'] == 1
    ]
    # The choices must correspond one-to-one with the conditions
    choices = ['21-35', '36-50', '51+', '<20']
    df['age_group'] = np.select(conditions, choices, default='Unknown')

# Now, prepare your feature and target datasets
X_train = train_data.drop(age_columns + ['age_group'], axis=1)
y_train = train_data['age_group']
X_val = val_data.drop(age_columns + ['age_group'], axis=1)
y_val = val_data['age_group']
X_test = test_data.drop(age_columns + ['age_group'], axis=1)
y_test = test_data['age_group']
```

```
from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier(random_state=11)
dt_classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=11)
```

```
from sklearn.tree import plot_tree

y_val_pred = dt_classifier.predict(X_val)

y_train_pred = dt_classifier.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)

val_accuracy = accuracy_score(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)
val_class_report = classification_report(y_val, y_val_pred)

print(f"Training Accuracy: {train_accuracy}")
print(f"Validation Accuracy: {val_accuracy}")
print("Confusion Matrix:\n", val_conf_matrix)
print("Classification Report:\n", val_class_report)

# This takes about 2 minutes
# plt.figure(figsize=(20,10))
# plot_tree(dt_classifier, feature_names=X_train.columns, class_names=age_groups, filled=Tr
```

```
Training Accuracy: 0.9996869129618033
Validation Accuracy: 0.38059467918622847
Confusion Matrix:
```

```
 [[557 340 200 142]
 [287 329 194  88]
 [202 205 241  53]
 [127  82  59  89]]
Classification Report:
               precision    recall  f1-score   support

       21-35       0.47      0.45      0.46      1239
       36-50       0.34      0.37      0.35       898
         51+       0.35      0.34      0.35       701
         <20       0.24      0.25      0.24       357

    accuracy                           0.38      3195
   macro avg       0.35      0.35      0.35      3195
weighted avg       0.38      0.38      0.38      3195
```

```python
y_test_pred = dt_classifier.predict(X_test)

test_accuracy = accuracy_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

print(f"Test Accuracy: {test_accuracy}")
print("Confusion Matrix on Test Data:\n", test_conf_matrix)
```

```
Test Accuracy: 0.38841940532081376
Confusion Matrix on Test Data:
 [[552 307 202 138]
 [301 356 197  85]
 [191 195 232  60]
 [132  86  60 101]]
```

1. Random Forests

```python
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(random_state=17)
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=17)
```

```python
y_val_pred = rf_classifier.predict(X_val)
y_train_pred = rf_classifier.predict(X_train)

# Evaluating the model
train_accuracy = accuracy_score(y_train, y_train_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)
val_conf_matrix = confusion_matrix(y_val, y_val_pred)
val_class_report = classification_report(y_val, y_val_pred)
```

```
print(f"Random Forest Training Accuracy: {train_accuracy}")
print(f"Validation Accuracy: {val_accuracy}")
print("Confusion Matrix:\n", val_conf_matrix)
print("Classification Report:\n", val_class_report)
```

```
Random Forest Training Accuracy: 0.9996869129618033
Validation Accuracy: 0.48012519561815337
Confusion Matrix:
 [[914 228  82  15]
 [433 338 117  10]
 [271 187 241   2]
 [243  58  15  41]]
Classification Report:
               precision    recall  f1-score   support

       21-35       0.49      0.74      0.59      1239
       36-50       0.42      0.38      0.40       898
         51+       0.53      0.34      0.42       701
         <20       0.60      0.11      0.19       357

    accuracy                           0.48      3195
   macro avg       0.51      0.39      0.40      3195
weighted avg       0.49      0.48      0.45      3195
```

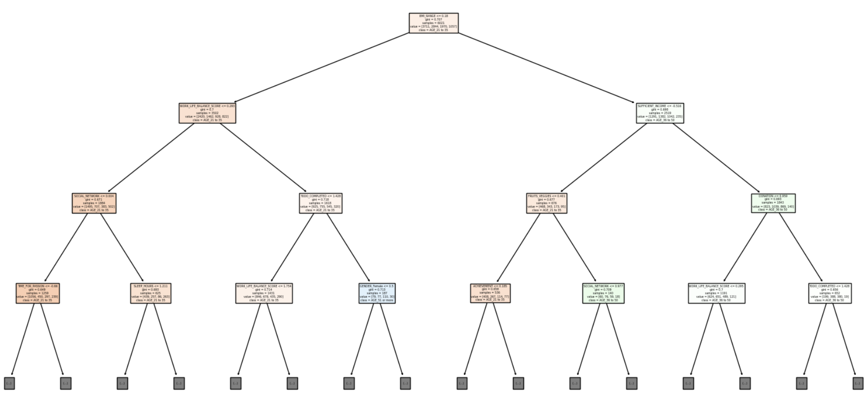
```
y_test_pred = rf_classifier.predict(X_test)

# Evaluating the model on the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

print(f"Test Accuracy: {test_accuracy}")
print("Confusion Matrix on Test Data:\n", test_conf_matrix)
```

```
Test Accuracy: 0.48200312989045385
Confusion Matrix on Test Data:
 [[897 208  74  20]
 [458 356 116   9]
 [258 192 227   1]
 [248  53  18  60]]
```

```
plt.figure(figsize=(20,10))
plot_tree(rf_classifier.estimators_[0], feature_names=X_train.columns, class_names=age_colu
```

```
[Text(0.5, 0.9, 'BMI_RANGE <= 0.18\ngini = 0.707\nsamples = 6021\nvalue = [3711, 2844, 1970,
 Text(0.25, 0.7, 'WORK_LIFE_BALANCE_SCORE <= 0.293\ngini = 0.7\nsamples = 3502\nvalue = [242
 Text(0.125, 0.5, 'SOCIAL_NETWORK <= 0.004\ngini = 0.671\nsamples = 1884\nvalue = [1495, 70
```

```
Text(0.0625, 0.3, 'TIME_FOR_PASSION <= -0.66\ngini = 0.649\nsamples = 1259\nvalue = [1056,
Text(0.03125, 0.1, '\n  (...)  \n'),
Text(0.09375, 0.1, '\n  (...)  \n'),
Text(0.1875, 0.3, 'SLEEP_HOURS <= 1.211\ngini = 0.693\nsamples = 625\nvalue = [439, 257, 86
Text(0.15625, 0.1, '\n  (...)  \n'),
Text(0.21875, 0.1, '\n  (...)  \n'),
Text(0.375, 0.5, 'TODO_COMPLETED <= 1.428\ngini = 0.718\nsamples = 1618\nvalue = [925, 755,
Text(0.3125, 0.3, 'WORK_LIFE_BALANCE_SCORE <= 1.754\ngini = 0.714\nsamples = 1431\nvalue =
Text(0.28125, 0.1, '\n  (...)  \n'),
Text(0.34375, 0.1, '\n  (...)  \n'),
Text(0.4375, 0.3, 'GENDER_Female <= 0.5\ngini = 0.713\nsamples = 187\nvalue = [79, 77, 110,
Text(0.40625, 0.1, '\n  (...)  \n'),
Text(0.46875, 0.1, '\n  (...)  \n'),
Text(0.75, 0.7, 'SUFFICIENT_INCOME <= -0.516\ngini = 0.698\nsamples = 2519\nvalue = [1291,
Text(0.625, 0.5, 'FRUITS_VEGGIES <= 0.401\ngini = 0.677\nsamples = 676\nvalue = [468, 343,
Text(0.5625, 0.3, 'ACHIEVEMENT <= 0.185\ngini = 0.658\nsamples = 536\nvalue = [408, 267, 1
Text(0.53125, 0.1, '\n  (...)  \n'),
Text(0.59375, 0.1, '\n  (...)  \n'),
Text(0.6875, 0.3, 'SOCIAL_NETWORK <= 0.977\ngini = 0.709\nsamples = 140\nvalue = [60, 76, 5
Text(0.65625, 0.1, '\n  (...)  \n'),
Text(0.71875, 0.1, '\n  (...)  \n'),
Text(0.875, 0.5, 'DONATION <= 0.959\ngini = 0.693\nsamples = 1843\nvalue = [823, 1039, 869,
Text(0.8125, 0.3, 'WORK_LIFE_BALANCE_SCORE <= 0.295\ngini = 0.7\nsamples = 1191\nvalue = [6
Text(0.78125, 0.1, '\n  (...)  \n'),
Text(0.84375, 0.1, '\n  (...)  \n'),
Text(0.9375, 0.3, 'TODO_COMPLETED <= 1.428\ngini = 0.656\nsamples = 652\nvalue = [199, 388,
Text(0.90625, 0.1, '\n  (...)  \n'),
Text(0.96875, 0.1, '\n  (...)  \n')]
```



1. Neural Networks

```python
# Reload the data
train_data = pd.read_csv("data_train.csv")
val_data = pd.read_csv("data_val.csv")
test_data = pd.read_csv("data_test.csv")

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
from sklearn.metrics import mean_squared_error

X_train = train_data.drop('WORK_LIFE_BALANCE_SCORE', axis=1)
y_train = train_data['WORK_LIFE_BALANCE_SCORE']
X_val = val_data.drop('WORK_LIFE_BALANCE_SCORE', axis=1)
y_val = val_data['WORK_LIFE_BALANCE_SCORE']
X_test = test_data.drop('WORK_LIFE_BALANCE_SCORE', axis=1)
y_test = test_data['WORK_LIFE_BALANCE_SCORE']

def build_model(input_shape):
    # Initialize the neural net structure
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1)
    ])
    # Compile the model
    model.compile(optimizer='adam', loss='mse')
    return model

num_seeds = 10
all_scoresT = []
all_scoresV = []

# n different seeds for training
for i in range(num_seeds):
    print(f"Training on seed {i}")
    tf.random.set_seed(i)
    model = build_model(X_train.shape[1])
    model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)

    y_train_pred = model.predict(X_train)
    train_score = np.sqrt(mean_squared_error(y_train, y_train_pred))
    all_scoresT.append(train_score)

    y_val_pred = model.predict(X_val)
    val_score = np.sqrt(mean_squared_error(y_val, y_val_pred))
    all_scoresV.append(val_score)
```

```python
average_scoreT = np.mean(all_scoresT)
average_scoreV = np.mean(all_scoresV)
print(f"Average Training Score: {average_scoreT}")
print(f"Average Validation Score: {average_scoreV}")
```

```
Training on seed 0
Epoch 1/10
300/300 [==============================] - 0s 538us/step - loss: 0.0573
Epoch 2/10
300/300 [==============================] - 0s 513us/step - loss: 0.0075
Epoch 3/10
300/300 [==============================] - 0s 517us/step - loss: 0.0042
Epoch 4/10
300/300 [==============================] - 0s 533us/step - loss: 0.0028
Epoch 5/10
300/300 [==============================] - 0s 540us/step - loss: 0.0020
Epoch 6/10
300/300 [==============================] - 0s 534us/step - loss: 0.0015
Epoch 7/10
300/300 [==============================] - 0s 535us/step - loss: 0.0012
Epoch 8/10
300/300 [==============================] - 0s 524us/step - loss: 9.6339e-04
Epoch 9/10
300/300 [==============================] - 0s 539us/step - loss: 8.2076e-04
Epoch 10/10
300/300 [==============================] - 0s 518us/step - loss: 6.6603e-04
300/300 [==============================] - 0s 336us/step
100/100 [==============================] - 0s 342us/step
Training on seed 1
Epoch 1/10
300/300 [==============================] - 0s 534us/step - loss: 0.0531
Epoch 2/10
300/300 [==============================] - 0s 562us/step - loss: 0.0074
Epoch 3/10
300/300 [==============================] - 0s 557us/step - loss: 0.0041
Epoch 4/10
300/300 [==============================] - 0s 557us/step - loss: 0.0028
Epoch 5/10
300/300 [==============================] - 0s 542us/step - loss: 0.0020
Epoch 6/10
300/300 [==============================] - 0s 539us/step - loss: 0.0016
Epoch 7/10
300/300 [==============================] - 0s 537us/step - loss: 0.0013
Epoch 8/10
300/300 [==============================] - 0s 539us/step - loss: 0.0010
Epoch 9/10
```

```
300/300 [==============================] - 0s 538us/step - loss: 8.5368e-04
Epoch 10/10
300/300 [==============================] - 0s 531us/step - loss: 6.9508e-04
300/300 [==============================] - 0s 349us/step
100/100 [==============================] - 0s 365us/step
Training on seed 2
Epoch 1/10
300/300 [==============================] - 0s 551us/step - loss: 0.0583
Epoch 2/10
300/300 [==============================] - 0s 540us/step - loss: 0.0069
Epoch 3/10
300/300 [==============================] - 0s 532us/step - loss: 0.0038
Epoch 4/10
300/300 [==============================] - 0s 535us/step - loss: 0.0025
Epoch 5/10
300/300 [==============================] - 0s 536us/step - loss: 0.0018
Epoch 6/10
300/300 [==============================] - 0s 531us/step - loss: 0.0014
Epoch 7/10
300/300 [==============================] - 0s 541us/step - loss: 0.0011
Epoch 8/10
300/300 [==============================] - 0s 535us/step - loss: 8.1736e-04
Epoch 9/10
300/300 [==============================] - 0s 524us/step - loss: 6.4278e-04
Epoch 10/10
300/300 [==============================] - 0s 520us/step - loss: 5.2958e-04
300/300 [==============================] - 0s 351us/step
100/100 [==============================] - 0s 367us/step
Training on seed 3
Epoch 1/10
300/300 [==============================] - 0s 555us/step - loss: 0.0562
Epoch 2/10
300/300 [==============================] - 0s 545us/step - loss: 0.0079
Epoch 3/10
300/300 [==============================] - 0s 543us/step - loss: 0.0047
Epoch 4/10
300/300 [==============================] - 0s 544us/step - loss: 0.0031
Epoch 5/10
300/300 [==============================] - 0s 539us/step - loss: 0.0024
Epoch 6/10
300/300 [==============================] - 0s 529us/step - loss: 0.0017
Epoch 7/10
300/300 [==============================] - 0s 531us/step - loss: 0.0014
Epoch 8/10
300/300 [==============================] - 0s 546us/step - loss: 0.0011
Epoch 9/10
```

```
300/300 [==============================] - 0s 546us/step - loss: 8.9771e-04
Epoch 10/10
300/300 [==============================] - 0s 544us/step - loss: 7.5501e-04
300/300 [==============================] - 0s 356us/step
100/100 [==============================] - 0s 356us/step
Training on seed 4
Epoch 1/10
300/300 [==============================] - 0s 560us/step - loss: 0.0668
Epoch 2/10
300/300 [==============================] - 0s 538us/step - loss: 0.0066
Epoch 3/10
300/300 [==============================] - 0s 536us/step - loss: 0.0037
Epoch 4/10
300/300 [==============================] - 0s 556us/step - loss: 0.0025
Epoch 5/10
300/300 [==============================] - 0s 528us/step - loss: 0.0018
Epoch 6/10
300/300 [==============================] - 0s 528us/step - loss: 0.0014
Epoch 7/10
300/300 [==============================] - 0s 541us/step - loss: 0.0010
Epoch 8/10
300/300 [==============================] - 0s 551us/step - loss: 8.5196e-04
Epoch 9/10
300/300 [==============================] - 0s 549us/step - loss: 7.2771e-04
Epoch 10/10
300/300 [==============================] - 0s 542us/step - loss: 5.6479e-04
300/300 [==============================] - 0s 358us/step
100/100 [==============================] - 0s 363us/step
Training on seed 5
Epoch 1/10
300/300 [==============================] - 0s 562us/step - loss: 0.0693
Epoch 2/10
300/300 [==============================] - 0s 594us/step - loss: 0.0080
Epoch 3/10
300/300 [==============================] - 0s 537us/step - loss: 0.0044
Epoch 4/10
300/300 [==============================] - 0s 533us/step - loss: 0.0030
Epoch 5/10
300/300 [==============================] - 0s 532us/step - loss: 0.0022
Epoch 6/10
300/300 [==============================] - 0s 538us/step - loss: 0.0017
Epoch 7/10
300/300 [==============================] - 0s 533us/step - loss: 0.0013
Epoch 8/10
300/300 [==============================] - 0s 533us/step - loss: 0.0011
Epoch 9/10
```

```
300/300 [==============================] - 0s 539us/step - loss: 8.4299e-04
Epoch 10/10
300/300 [==============================] - 0s 540us/step - loss: 6.9731e-04
300/300 [==============================] - 0s 359us/step
100/100 [==============================] - 0s 350us/step
Training on seed 6
Epoch 1/10
300/300 [==============================] - 0s 551us/step - loss: 0.0636
Epoch 2/10
300/300 [==============================] - 0s 551us/step - loss: 0.0086
Epoch 3/10
300/300 [==============================] - 0s 535us/step - loss: 0.0048
Epoch 4/10
300/300 [==============================] - 0s 538us/step - loss: 0.0032
Epoch 5/10
300/300 [==============================] - 0s 535us/step - loss: 0.0023
Epoch 6/10
300/300 [==============================] - 0s 535us/step - loss: 0.0017
Epoch 7/10
300/300 [==============================] - 0s 535us/step - loss: 0.0014
Epoch 8/10
300/300 [==============================] - 0s 531us/step - loss: 0.0011
Epoch 9/10
300/300 [==============================] - 0s 533us/step - loss: 8.6700e-04
Epoch 10/10
300/300 [==============================] - 0s 560us/step - loss: 7.0429e-04
300/300 [==============================] - 0s 357us/step
100/100 [==============================] - 0s 366us/step
Training on seed 7
Epoch 1/10
300/300 [==============================] - 0s 544us/step - loss: 0.0741
Epoch 2/10
300/300 [==============================] - 0s 548us/step - loss: 0.0086
Epoch 3/10
300/300 [==============================] - 0s 531us/step - loss: 0.0051
Epoch 4/10
300/300 [==============================] - 0s 541us/step - loss: 0.0035
Epoch 5/10
300/300 [==============================] - 0s 561us/step - loss: 0.0025
Epoch 6/10
300/300 [==============================] - 0s 546us/step - loss: 0.0020
Epoch 7/10
300/300 [==============================] - 0s 540us/step - loss: 0.0016
Epoch 8/10
300/300 [==============================] - 0s 539us/step - loss: 0.0013
Epoch 9/10
```

```
300/300 [==============================] - 0s 537us/step - loss: 0.0010
Epoch 10/10
300/300 [==============================] - 0s 545us/step - loss: 9.0563e-04
300/300 [==============================] - 0s 346us/step
100/100 [==============================] - 0s 368us/step
Training on seed 8
Epoch 1/10
300/300 [==============================] - 0s 573us/step - loss: 0.0721
Epoch 2/10
300/300 [==============================] - 0s 537us/step - loss: 0.0088
Epoch 3/10
300/300 [==============================] - 0s 533us/step - loss: 0.0051
Epoch 4/10
300/300 [==============================] - 0s 526us/step - loss: 0.0034
Epoch 5/10
300/300 [==============================] - 0s 530us/step - loss: 0.0024
Epoch 6/10
300/300 [==============================] - 0s 527us/step - loss: 0.0019
Epoch 7/10
300/300 [==============================] - 0s 527us/step - loss: 0.0015
Epoch 8/10
300/300 [==============================] - 0s 532us/step - loss: 0.0012
Epoch 9/10
300/300 [==============================] - 0s 530us/step - loss: 0.0010
Epoch 10/10
300/300 [==============================] - 0s 529us/step - loss: 8.3641e-04
300/300 [==============================] - 0s 342us/step
100/100 [==============================] - 0s 352us/step
Training on seed 9
Epoch 1/10
300/300 [==============================] - 0s 577us/step - loss: 0.0569
Epoch 2/10
300/300 [==============================] - 0s 570us/step - loss: 0.0079
Epoch 3/10
300/300 [==============================] - 0s 545us/step - loss: 0.0045
Epoch 4/10
300/300 [==============================] - 0s 571us/step - loss: 0.0029
Epoch 5/10
300/300 [==============================] - 0s 536us/step - loss: 0.0020
Epoch 6/10
300/300 [==============================] - 0s 536us/step - loss: 0.0015
Epoch 7/10
300/300 [==============================] - 0s 996us/step - loss: 0.0012
Epoch 8/10
300/300 [==============================] - 1s 2ms/step - loss: 9.4196e-04
Epoch 9/10
```

```
300/300 [==============================] - 0s 1ms/step - loss: 7.6574e-04
Epoch 10/10
300/300 [==============================] - 0s 845us/step - loss: 6.3096e-04
300/300 [==============================] - 0s 356us/step
100/100 [==============================] - 0s 376us/step
Average Training Score: 0.024622411715387073
Average Validation Score: 0.030026559600401764
```

```python
# Test set
y_test_pred = model.predict(X_test)
test_score = np.sqrt(mean_squared_error(y_test, y_test_pred))
print(f"Test Score: {test_score}")
```

```
100/100 [==============================] - 0s 353us/step
Test Score: 0.02799366725417523
```