- AI-Based Wellness Scoring System Framework
  - 1. Data Collection and Preprocessing
    - Objective
    - Steps
      - Aggregate Data
      - Data Cleaning
      - Normalization
      - Categorical Encoding
    - Example Workflow
  - 2. Weighted Scoring Calculation
    - Objective
    - Steps
      - Define Weights
      - Calculate Individual Scores
      - Aggregate Weighted Scores
    - Example Workflow
  - 3. Machine Learning for Predictive Scoring (Hugging Face)
    - Objective
    - Steps
  - 4. Family and Community-Level Adjustments
    - Objective
    - Steps
      - Family Score Calculation
      - Community Score Calculation
    - Example Workflow
  - 5. Continuous Learning and Adaptation
    - Objective
    - Steps
      - Retraining the Model
      - Adaptive Weight Adjustment
      - User Feedback Integration
    - Example Workflow
  - 6. Score Output and Reporting
    - Objective
    - Steps
      - Store the Calculated Scores
      - Generate Reports

- Example Workflow

AI-Based Wellness Scoring System Framework

This document outlines the framework for calculating wellness scores using AI techniques, focusing on data collection, weighted scoring, predictive modeling, and family and community-level adjustments.

---

1. Data Collection and Preprocessing

Objective

Aggregate and prepare data from various sources (Health Database, Life Event Database, Social Trend Database, etc.) to create a unified dataset for calculating wellness scores.

Steps

Aggregate Data

- **Collect Data from Relevant Sources**:
  - **Health Data**: Retrieve metrics such as heart rate, sleep quality, activity levels from the **Health Database**.
  - **Life Events**: Gather records of significant events (e.g., job changes, hospital visits) from the **Life Event Database**.
  - **Social Trends and Urban Issues**: Incorporate broader social factors like unemployment rates, crime statistics, and access to healthcare from the **Social Trend Database** and **Urban Issue Database**.
- **Merge Data**: Combine individual, family, and community data sources into a single dataset. This allows for comprehensive analysis at different levels.
  - *Python Packages*: `pandas` for data merging and manipulation
  - *Example*:

```
import pandas as pd

# Load data from CSV or database sources
health_data = pd.read_csv("health_data.csv")
life_event_data = pd.read_csv("life_event_data.csv")
social_trend_data = pd.read_csv("social_trend_data.csv")

# Merge datasets on common identifiers (e.g., user_id, timestamp)
combined_data = pd.merge(health_data, life_event_data, on="user_id")
combined_data = pd.merge(combined_data, social_trend_data,
on="community_id")
```

Data Cleaning

- **Handle Missing Values**: Use methods such as mean/median imputation for continuous variables and mode imputation for categorical variables. Alternatively, drop rows or columns with excessive missing data.
  - *Python Packages*: `pandas` for filling or dropping missing data
  - *Example*:

```
# Fill missing values with column mean or median
combined_data['heart_rate'].fillna(combined_data['heart_rate'].mean(),
inplace=True)
combined_data['event_type'].fillna('Unknown', inplace=True)
```

- **Outlier Detection and Handling**: Identify and handle outliers, especially in continuous variables like heart rate and sleep duration, using z-score or IQR methods.
  - *Python Packages*: `numpy`, `scipy.stats`
  - *Example*:

```
from scipy.stats import zscore

# Filter out outliers using z-score threshold (e.g., z > 3)
combined_data = combined_data[(zscore(combined_data['heart_rate']) < 3)]
```

Normalization

- **Standardize Continuous Variables**: Scale data to a common range (e.g., 0 to 1) or standardize to zero mean and unit variance to improve model performance.

- *Python Packages*: `sklearn.preprocessing` (e.g., `StandardScaler`, `MinMaxScaler`)
- *Example*:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Apply StandardScaler to health metrics
scaler = StandardScaler()
combined_data[['heart_rate', 'sleep_quality', 'activity_level']] =
scaler.fit_transform(
    combined_data[['heart_rate', 'sleep_quality', 'activity_level']]
)
```

Categorical Encoding

- **Convert Categorical Data**: Transform categorical variables such as event types (e.g., "job change", "hospital visit") into numerical formats using one-hot encoding or embeddings. This makes them suitable for machine learning models.
  - *Python Packages*: `sklearn.preprocessing` (e.g., `OneHotEncoder`)
  - *Example*:

```
from sklearn.preprocessing import OneHotEncoder

# Use OneHotEncoder to convert event_type column to binary vectors
encoder = OneHotEncoder(sparse=False)
event_type_encoded = encoder.fit_transform(combined_data[['event_type']])

# Create a new DataFrame from the encoded data and join with
combined_data
event_type_df = pd.DataFrame(event_type_encoded,
columns=encoder.get_feature_names_out(['event_type']))
combined_data = pd.concat([combined_data, event_type_df], axis=1)
```

Example Workflow

1. **Load Data**: Retrieve data from multiple sources and merge into a single dataset.
2. **Clean Data**: Fill missing values, remove outliers, and handle inconsistencies.
3. **Normalize Continuous Variables**: Scale health metrics and other continuous data to standardize the dataset.

4. **Encode Categorical Data**: Transform categorical features to numerical representations through one-hot encoding.
5. **Output Preprocessed Data**: The final preprocessed dataset is ready for use in the wellness scoring algorithm.

This approach ensures that the dataset is clean, consistent, and prepared for further analysis, facilitating accurate wellness score calculations.

---

2. Weighted Scoring Calculation

Objective

Apply weights to different categories of inputs (e.g., health data, life events, social trends) to calculate an initial wellness score. This allows the system to reflect the relative importance of each category in the overall wellness assessment.

Steps

Define Weights

- **Assign Predefined Weights**: Determine the relative importance of each category (e.g., health data, life events, social trends) by assigning weights that sum to 1. For example:
  - Health Data: **0.6** (reflecting its direct impact on individual wellness)
  - Life Events: **0.3** (accounting for significant changes in personal circumstances)
  - Social Trends: **0.1** (representing broader community or societal influences)
- **Customize Weights as Needed**: Weights can be adjusted based on specific project requirements or recalibrated over time based on feedback and model performance.

Calculate Individual Scores

- **Apply Category Weights**: For each data point, multiply its value by the corresponding weight to calculate its contribution to the wellness score.
  - Example Calculation:

```
# Assume the following values
health_score = 0.8
life_event_score = 0.5
social_trend_score = 0.3

# Assign weights
health_weight = 0.6
life_event_weight = 0.3
social_trend_weight = 0.1

# Calculate weighted scores
weighted_health_score = health_score * health_weight  # 0.8 * 0.6 = 0.48
weighted_life_event_score = life_event_score * life_event_weight  # 0.5 * 0.3 = 0.15
weighted_social_trend_score = social_trend_score * social_trend_weight  # 0.3 * 0.1 = 0.03
```

Aggregate Weighted Scores

- **Sum the Weighted Scores**: Combine the weighted scores from each category to produce the initial wellness score. This score reflects the overall wellness based on the specified weights for each category.

  - Formula:

```
initial_wellness_score = weighted_health_score + weighted_life_event_score + weighted_social_trend_score
```

  - Using the example above:

```
initial_wellness_score = 0.48 + 0.15 + 0.03  # Result: 0.66
```

- *Python Packages*: `numpy` for vectorized operations, which can simplify calculations when handling multiple scores.

  - Example:

```python
import numpy as np

# Define score and weight arrays
scores = np.array([health_score, life_event_score, social_trend_score])
weights = np.array([health_weight, life_event_weight,
social_trend_weight])

# Calculate initial wellness score as a dot product
initial_wellness_score = np.dot(scores, weights)
```

Example Workflow

1. **Define Category Weights**: Establish the weights for health, life events, and social trends based on their importance to overall wellness.
2. **Calculate Weighted Scores**: Multiply each category score by its weight to get weighted contributions.
3. **Aggregate to Obtain Initial Wellness Score**: Sum the weighted scores to calculate the initial wellness score.
4. **Output Initial Score**: The resulting initial wellness score represents the weighted influence of each category on overall well-being.

This approach provides a straightforward calculation of the initial wellness score, which serves as a baseline for further refinement through machine learning and adaptive feedback.

---

3. Machine Learning for Predictive Scoring (Hugging Face)

Objective

Use a machine learning model from Hugging Face to refine and predict the wellness score based on current inputs.

Steps

- **Load Pre-Trained Model**: Utilize a pre-trained transformer model from Hugging Face that can be fine-tuned for regression tasks. For example, BERT-based models like `bert-base-uncased` can be adapted for regression by modifying the output layer.
- *Python Packages*: `transformers` from Hugging Face
- Example:

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer

model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=1)  # For regression
```

- **Fine-Tune the Model**: Fine-tune the model on historical wellness score data to adapt it to the task of predicting wellness scores based on life events and health data. This step requires labeled training data where inputs (event and health descriptions) are paired with their corresponding wellness scores.
- *Python Packages*: `transformers`, `datasets`
- Example:

```
from datasets import load_dataset

dataset = load_dataset('your_dataset')  # Replace with your dataset
# Fine-tuning loop or trainer
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results', num_train_epochs=3, per_device_train_batch_size=4,
    warmup_steps=500, weight_decay=0.01, logging_dir='./logs'
)

trainer = Trainer(
    model=model, args=training_args, train_dataset=dataset["train"],
eval_dataset=dataset["test"]
)
trainer.train()
```

- **Predictive Adjustment**: Use the fine-tuned model to predict the impact of new data points on the wellness score. This model can take in text inputs describing life events and health conditions to output a score prediction.

- Example:

```
inputs = tokenizer("User had a high-stress event and poor sleep quality",
return_tensors="pt")
predicted_score = model(**inputs).logits.item()
```

- **Combine Predictions with Weighted Scores**: Adjust the initial wellness score by averaging it with the predictive model' s output: Refined Wellness Score = (Initial Wellness Score + Predicted Score) / 2

---

4. Family and Community-Level Adjustments

Objective

Aggregate individual wellness scores to produce family and community-level scores, adjusting for broader trends such as social and economic conditions within the community.

Steps

Family Score Calculation

- **Aggregate Individual Scores**: Calculate the family wellness score by averaging the wellness scores of all family members. This provides a measure of overall family well-being based on individual scores.
  - Formula:

    ```
    Family Score = (Sum of Individual Wellness Scores) / (Number of Family
    Members)
    ```

- **Adjust for Family Events**: Incorporate any family-level events that impact the wellness score (e.g., a family-wide illness or financial issue). Apply an adjustment factor based on the severity of these events.

- Example:

```
Family Score = (Sum of Individual Wellness Scores) / (Number of Family
Members) - Family Event Adjustment
```

- *Python Packages*: `numpy` for calculating averages and adjustments, `pandas` for organizing data by family.

Community Score Calculation

- **Aggregate Family Scores**: Calculate the community wellness score by averaging the scores of all families within the community. This helps capture the overall well-being of the community as a whole.

  - Formula:

  ```
  Community Score = (Sum of Family Scores) / (Number of Families)
  ```

- **Incorporate Social Trends**: Adjust the community score based on social trends and broader community events (e.g., high unemployment, crime rates, or healthcare access). These adjustments account for external factors that impact the community’ s well-being.

  - Formula:

  ```
  Community Score = (Sum of Family Scores) / (Number of Families) + Social
  Trend Adjustment
  ```

  - *Example*: If there is a 5% increase in unemployment in the community, reduce the community score by a factor based on this trend.

- **Apply Weighting Based on Trend Severity**: Assign weights to different social trends based on their impact severity (e.g., health trends may be weighted more heavily than economic trends).

  - Final Formula:

```
Community Score = (Sum of Family Scores) / (Number of Families) +
(Trend_1 * Weight_1) + (Trend_2 * Weight_2) + ...
```

- *Python Packages*:
  - `numpy` for averaging and applying weighted adjustments.
  - `pandas` for data manipulation, organizing family scores, and calculating community-level metrics.
  - `sklearn.preprocessing` for scaling and normalizing social trend data if needed.

Example Workflow

1. **Load Data**: Retrieve individual scores for each family from the User Score Database, and calculate family scores by averaging individual wellness scores.
2. **Apply Family-Level Adjustments**: Modify the family scores based on any significant family-level events.
3. **Calculate Community Score**: Aggregate family scores to calculate the average community score.
4. **Adjust for Social Trends**: Integrate social trend data and adjust the community score accordingly. For example, if there is an increase in local crime rates, decrease the community score proportionally.
5. **Output Scores**: Store the final family and community scores in the Family Score Database and Community Score Database for future analysis and reporting.

This approach ensures that the wellness scoring framework reflects both individual and collective factors that influence well-being, accounting for personal, family, and community-level data.

---

5. Continuous Learning and Adaptation

Objective

Implement a feedback loop that allows the scoring model to continuously learn and adapt based on new data, trends, and user feedback. This ensures that the wellness scoring model remains relevant and accurate over time.

Steps

Retraining the Model

- **Regular Retraining**: Periodically retrain the Hugging Face model using updated data to capture new trends and patterns in wellness scores. This involves collecting new labeled data on wellness scores, health metrics, and life events.
- **Saving and Loading the Model**: Use Hugging Face’s `save_pretrained` and `from_pretrained` methods to save the model after training and load it for future sessions.
    - Example:

    ```
    # Saving the model after retraining
    model.save_pretrained('./model_checkpoint')
    tokenizer.save_pretrained('./model_checkpoint')

    # Loading the saved model for continued training or inference
    from transformers import AutoModelForSequenceClassification,
    AutoTokenizer

    model =
    AutoModelForSequenceClassification.from_pretrained('./model_checkpoint')
    tokenizer = AutoTokenizer.from_pretrained('./model_checkpoint')
    ```

- **Fine-Tuning with New Data**: After loading, continue fine-tuning the model on new data to ensure it incorporates recent trends in life events and health metrics.
    - *Python Packages*: `transformers` for model management, `datasets` for handling new data

Adaptive Weight Adjustment

- **Dynamic Weight Adjustment**: Adjust the weights of different categories (e.g., health, life events, social trends) based on recent model performance. If certain categories consistently correlate more strongly with changes in wellness scores, increase their weights.
  - **Example Workflow**:
    1. Calculate the correlation between each category and the overall wellness score.
    2. Adjust the weights based on correlation strength; for instance, if health metrics have shown a stronger relationship with the wellness score, increase the weight assigned to health.
  - **Implementation**: Use a moving average or exponential smoothing technique to update weights gradually over time, avoiding drastic changes that could disrupt model stability.
    - *Python Packages*: `pandas` for calculating moving averages, `numpy` for mathematical operations

User Feedback Integration

- **Collect User Feedback**: Allow users to provide feedback on the relevance and accuracy of their wellness scores. This could be done through a rating system (e.g., 1 to 5 stars) or a brief questionnaire about the score's accuracy.
- **Incorporate Feedback into Model**: Use the feedback as part of a retraining dataset or to modify scoring thresholds dynamically. For instance, if a significant number of users indicate that their scores feel inaccurate, adjust the weights or retrain the model with feedback-labeled data.
- **Example**:
  - Aggregate user feedback, and if patterns emerge (e.g., lower feedback ratings on scores linked to social trends), recalibrate the weights or the scoring mechanism for that category.
- **Feedback Loop for Continual Improvement**: Set up a regular schedule for analyzing user feedback and adjusting the model accordingly. This helps the model to not only reflect quantitative data but also qualitative user insights over time.
  - *Python Packages*: `pandas` for handling user feedback data, `scipy.stats` for statistical analysis of feedback trends

Example Workflow

1. **Collect New Data**: Regularly gather new data from health metrics, life events, and user feedback.
2. **Retrain Model**: Use the updated dataset to retrain the Hugging Face model periodically, adapting to new patterns in the data.
3. **Adjust Weights**: Dynamically adjust the category weights based on recent correlations between inputs and wellness scores.
4. **Incorporate User Feedback**: Integrate feedback as part of the model's training data or directly into scoring adjustments, creating a feedback loop that fine-tunes the model.
5. **Deploy Updated Model**: Save and deploy the newly trained model to ensure users receive the most accurate and up-to-date wellness scores.

By implementing this feedback loop, the wellness scoring system remains adaptive, responsive, and increasingly accurate over time, reflecting the latest trends and user needs.

---

6. Score Output and Reporting

Objective

Output the final wellness scores at individual, family, and community levels. Provide visualization tools to track wellness trends over time, enabling users and city officials to understand well-being dynamics and identify areas for intervention.

Steps

Store the Calculated Scores

- **Save Scores to Databases**: Write the final wellness scores to designated databases for individual, family, and community levels.
  - **Individual Scores**: Store each user's final wellness score in the **User Score Database**.
  - **Family Scores**: Aggregate individual scores to calculate the family score and store it in the **Family Score Database**.

- **Community Scores**: Aggregate family scores to calculate the community score and store it in the **Community Score Database**.
- *Example*:

```
import pandas as pd

# Example individual score storage
user_score_data = {
    'user_id': [1, 2, 3],
    'wellness_score': [0.75, 0.82, 0.66],
    'timestamp': ['2024-10-01', '2024-10-01', '2024-10-01']
}
user_score_df = pd.DataFrame(user_score_data)
user_score_df.to_csv('user_score_database.csv', mode='a', index=False,
header=False)
```

- *Python Packages*: pandas for data storage in CSV or database formats

Generate Reports

- **Visualize Wellness Scores**: Create visualizations of wellness scores over time for trend analysis. This allows users and city officials to monitor wellness changes at individual, family, and community levels.
  - **Time-Series Analysis**: Plot wellness scores over time using line charts to show how individual, family, and community wellness has evolved.
    - *Example*:

```
import matplotlib.pyplot as plt

# Example data for plotting
timestamps = ['2024-09-01', '2024-09-15', '2024-10-01']
scores = [0.70, 0.75, 0.78]

plt.plot(timestamps, scores, marker='o')
plt.title('Wellness Score Over Time')
plt.xlabel('Date')
plt.ylabel('Wellness Score')
plt.show()
```

  - **Heatmaps for Community Trends**: Use heatmaps to illustrate community wellness scores across different neighborhoods or regions. This can reveal areas with low wellness scores, helping city officials target interventions.

- *Example*:

```
import seaborn as sns
import numpy as np

# Example community score data for heatmap
community_scores = np.array([[0.7, 0.6, 0.8], [0.5, 0.65, 0.75],
[0.6, 0.7, 0.68]])

sns.heatmap(community_scores, annot=True, cmap="coolwarm")
plt.title('Community Wellness Scores by Region')
plt.show()
```

- **Bar Charts for Category Impact**: Create bar charts to show the impact of different categories (health, life events, social trends) on the wellness score. This helps visualize which categories are most influential for individual and community wellness.

- **Export Reports**: Save generated reports and visualizations as images or PDFs for easy sharing. This is particularly useful for city officials who may need to present findings.

  - *Python Packages*: `matplotlib` and `seaborn` for visualizations, `pdfkit` or `matplotlib` for saving charts as images/PDFs

Example Workflow

1. **Calculate Scores**: After calculating the final wellness scores for individuals, families, and communities, store each score in its respective database.
2. **Generate Visualizations**: Create visualizations such as line charts for time-series trends, heatmaps for community scores, and bar charts for category impacts.
3. **Export and Share**: Save visualizations to files and generate PDF reports summarizing the wellness trends over time. Share these reports with users and city officials as needed.
4. **Automate Reporting**: Schedule automated report generation (e.g., monthly or quarterly) to provide regular updates on wellness trends.

By implementing this approach, the wellness scoring framework not only calculates and stores wellness scores but also offers visual insights that help users and city officials make informed decisions. The visualizations aid in identifying trends and areas needing attention, supporting proactive wellness management and community interventions.
