Model Development Phase

Project Name : Covid - 19 Infant growth Analysis and Prediction

Model Selection Report

In this project, multiple machine learning models were evaluated for predicting infant developmental outcomes. Factors such as accuracy, confusion matrix, and classification report were considered to select the most suitable model.

Model	Description
Model - 1 : TabPFNClassifier	• A transformer-based probabilistic neural network designed for tabular data. • Outperformed Logistic Regression in accuracy and robustness. • Selected as the final model.
Model- 2 : XGBClassifier	• A gradient boosting–based ensemble model that builds multiple weak learners (decision trees) sequentially. • Handles both numerical and categorical features efficiently, with built-in regularization to prevent overfitting. • Achieved strong predictive performance

	and interpretability through feature importance analysis. • Considered as an alternative advanced model alongside TabPFNClassifier.

Model Development Steps :

Model	**Description**
Model - 1: Dataset Loading and Preprocessing	Loaded Infant_development_dataset.csv, handled missing values (mean for numeric, mode for categorical), and applied Label encoder for categorical features and target labels.

```
import numpy as np
import pandas as pd
```

```
dataset = pd.read_csv('infant_development_dataset.csv')
dataset.head()
```

	infant_id	age_months	height_cm	weight_kg	milestone_score	speech_score	period
0	1	31	87.131229	15.069140	68.021936	102.133267	pre_covid
1	2	25	87.237638	16.034869	75.305256	89.340480	pre_covid
2	3	23	85.887134	16.977430	55.608942	89.827530	pre_covid
3	4	30	85.666237	14.464047	81.424646	84.722783	pre_covid
4	5	5	73.751033	10.164924	77.066009	84.761291	pre_covid

Next steps: Generate code with dataset | New interactive sheet

```
dataset = dataset.sample(frac=1, random_state=42).reset_index(drop=True)
print(dataset.head())
```

```
   infant_id  age_months  height_cm  weight_kg  milestone_score  speech_score  \
0        111          17  76.769686  13.610782        85.009172     77.179558
1        420          26  83.051313  10.863639        68.045308     93.570985
2        566          28  81.182053  15.418023        91.281495     85.559252
3         78          13  75.995788  11.470351        75.280003     86.737652
4        182          26  82.293630  15.004258        85.510677     85.311021

      period
0  pre_covid
1  post_covid
2  post_covid
3  pre_covid
4  pre_covid
```

```
dataset.isna().sum()
```

	0
infant_id	0
age_months	0
height_cm	0
weight_kg	0
milestone_score	0
speech_score	0
period	0

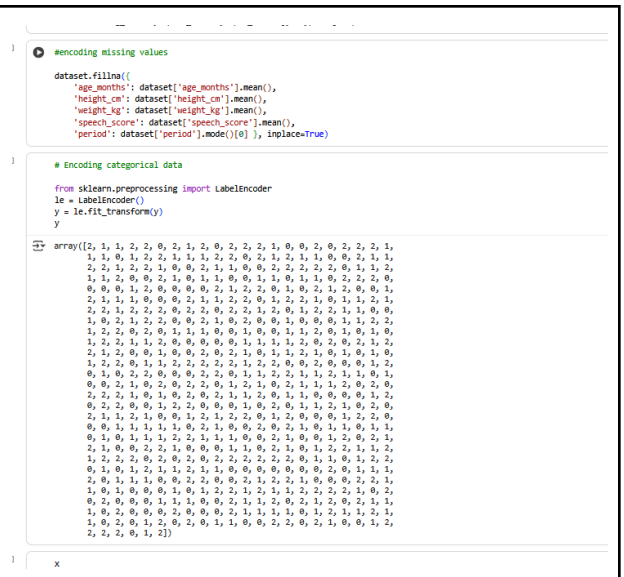
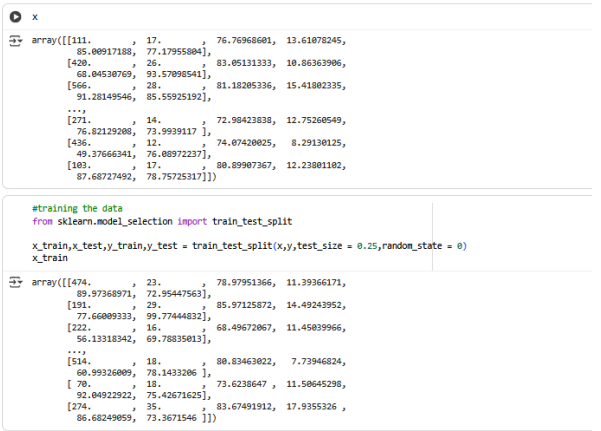
dtype: int64

```
x = dataset.iloc[:, :-1].values
x
```

```
array([[111.        ,  17.        ,  76.76968601,  13.61078245,
         85.00917188,  77.17955804],
       [420.        ,  26.        ,  83.05131333,  10.86363906,
         68.04530769,  93.57098541],
       [566.        ,  28.        ,  81.18205336,  15.41802335,
         91.28149546,  85.55925192],
       ...,
       [271.        ,  14.        ,  72.98423838,  12.75260549,
         76.82129208,  73.9939117 ],
       [436.        ,  12.        ,  74.07420025,   8.29130125,
         49.37666341,  76.08972237],
       [103.        ,  17.        ,  80.89907367,  12.23801102,
         87.68727492,  78.75725317]])
```

```
y = dataset.iloc[:, -1].values
y
```

```
'post_covid', 'pre_covid', 'pre_covid', 'during_covid',
'post_covid', 'pre_covid', 'during_covid', 'during_covid',
'during_covid', 'post_covid', 'pre_covid', 'pre_covid',
'during_covid', 'during_covid', 'during_covid', 'post_covid',
'post_covid', 'post_covid', 'post_covid', 'post_covid',
'during_covid', 'pre_covid', 'post_covid', 'during_covid',
'during_covid', 'pre_covid', 'during_covid', 'pre_covid',
'post_covid', 'during_covid', 'post_covid', 'post_covid',
'during_covid', 'post_covid', 'post_covid', 'during_covid',
'post_covid', 'during_covid', 'post_covid', 'post_covid',
'post_covid', 'pre_covid', 'pre_covid', 'post_covid', 'post_covid',
'post_covid', 'during_covid', 'during_covid', 'pre_covid',
'post_covid', 'during_covid', 'during_covid', 'post_covid',
'pre_covid', 'during_covid', 'pre_covid', 'post_covid',
'pre_covid', 'post_covid', 'during_covid', 'during_covid',
'pre_covid', 'pre_covid', 'post_covid', 'during_covid',
'during_covid', 'during_covid', 'post_covid', 'post_covid',
'during_covid', 'pre_covid', 'post_covid', 'during_covid',
'post_covid', 'pre_covid', 'pre_covid', 'post_covid', 'post_covid',
'pre_covid', 'post_covid', 'pre_covid', 'pre_covid', 'pre_covid',
'during_covid', 'pre_covid', 'during_covid', 'pre_covid',
'during_covid', 'pre_covid', 'pre_covid', 'pre_covid', 'pre_covid',
'pre_covid', 'pre_covid', 'during_covid', 'post_covid',
'post_covid', 'during_covid', 'post_covid', 'pre_covid',
'pre_covid', 'during_covid', 'post_covid', 'during_covid',
'post_covid', 'pre_covid', 'post_covid', 'post_covid', 'pre_covid',
'post_covid', 'post_covid', 'during_covid', 'during_covid',
'during_covid', 'during_covid', 'during_covid', 'during_covid',
'during_covid', 'pre_covid', 'during_covid', 'post_covid',
'post_covid', 'post_covid', 'pre_covid', 'during_covid',
'post_covid', 'post_covid', 'post_covid', 'during_covid',
'during_covid', 'pre_covid', 'pre_covid', 'during_covid',
'during_covid', 'pre_covid', 'post_covid', 'pre_covid',
'pre_covid', 'post_covid', 'during_covid', 'during_covid',
'during_covid', 'pre_covid', 'pre_covid', 'post_covid',
'post_covid', 'during_covid', 'post_covid', 'during_covid',
'during_covid', 'during_covid', 'post_covid', 'during_covid',
'post_covid', 'pre_covid', 'pre_covid', 'post_covid', 'pre_covid',
'post_covid', 'post_covid', 'pre_covid', 'pre_covid', 'pre_covid',
```

	
Model - 2 : Splitting dataset	Split the dataset into training (75%) and testing (25%) 
Model - 3: Feature Encoding	Encoded categorical variables (period) and the target class labels using Label encoder.

```
#encoding missing values

dataset.fillna({
    'age_months': dataset['age_months'].mean(),
    'height_cm': dataset['height_cm'].mean(),
    'weight_kg': dataset['weight_kg'].mean(),
    'speech_score': dataset['speech_score'].mean(),
    'period': dataset['period'].mode()[0] }, inplace=True)
```

```
# Encoding categorical data

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y
```

```
array([2, 1, 1, 2, 2, 0, 2, 1, 2, 0, 2, 2, 2, 1, 0, 0, 2, 0, 2, 2, 2, 1,
       1, 1, 0, 1, 2, 2, 1, 1, 1, 2, 2, 0, 2, 1, 2, 1, 1, 0, 0, 2, 1, 1,
       2, 2, 1, 2, 2, 1, 0, 0, 2, 1, 1, 0, 0, 2, 2, 2, 2, 2, 0, 1, 1, 2,
       1, 1, 2, 0, 0, 2, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 2, 2, 2, 0,
       0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 2, 1, 2, 2, 0, 1, 0, 2, 1, 2, 0, 0, 1,
       2, 1, 1, 1, 0, 0, 0, 2, 1, 1, 2, 2, 0, 1, 1, 2, 2, 1, 0, 1, 1, 2, 1,
       2, 2, 1, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 1, 2, 0, 1, 2, 2, 1, 1, 0, 0,
       1, 0, 2, 1, 2, 2, 0, 0, 2, 1, 0, 2, 0, 0, 1, 0, 0, 0, 1, 1, 2, 2,
       1, 2, 2, 0, 2, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 2, 0, 1, 0, 1, 0,
       1, 2, 2, 1, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 0, 2, 2, 1, 2,
       2, 1, 2, 0, 0, 1, 0, 0, 2, 0, 2, 1, 0, 1, 1, 2, 1, 0, 1, 0, 1, 0,
       1, 2, 2, 0, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 0, 0, 2, 0, 0, 0, 1, 2,
       0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 1, 1, 2, 2, 1, 1, 2, 1, 1, 0, 1,
       0, 0, 2, 1, 0, 2, 0, 2, 2, 0, 1, 2, 1, 0, 2, 1, 1, 1, 2, 0, 2, 0,
       2, 2, 2, 1, 0, 1, 0, 2, 0, 2, 1, 1, 2, 0, 1, 1, 0, 0, 0, 1, 2,
       0, 2, 2, 0, 0, 1, 2, 2, 0, 0, 0, 1, 0, 2, 0, 1, 2, 1, 0, 2, 0,
       2, 1, 1, 2, 1, 0, 0, 1, 2, 1, 2, 2, 0, 1, 2, 0, 0, 0, 1, 2, 2, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 2, 1, 0, 0, 2, 0, 2, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 0, 0, 2, 1, 0, 0, 1, 2, 0, 2, 1,
       2, 1, 0, 0, 2, 2, 1, 0, 0, 0, 1, 1, 0, 2, 1, 0, 1, 2, 2, 1, 1, 2,
       1, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 0, 1, 2, 2,
       0, 1, 0, 1, 2, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 1, 1,
       2, 0, 1, 1, 1, 0, 0, 2, 2, 0, 0, 2, 1, 2, 2, 1, 0, 0, 0, 2, 2, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 0, 2,
       0, 2, 0, 0, 0, 1, 1, 1, 0, 0, 2, 1, 1, 2, 0, 2, 1, 2, 0, 2, 2, 1, 1,
       1, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 1, 1, 1, 1, 0, 1, 2, 1, 1, 2, 1,
       1, 0, 2, 0, 1, 2, 0, 2, 0, 1, 1, 0, 0, 2, 2, 0, 2, 1, 0, 0, 1, 2,
       2, 2, 2, 0, 1, 2])
```

```
x
```

Model - 4 : Training the models

Trained XGBClassifier (baseline) and TabPFNClassifier (final model).

```
from tabpfn import TabPFNClassifier
```

```
tabpfn = TabPFNClassifier()
tabpfn.fit(x_train, y_train)
```

```
TabPFNClassifier
TabPFNClassifier()
```

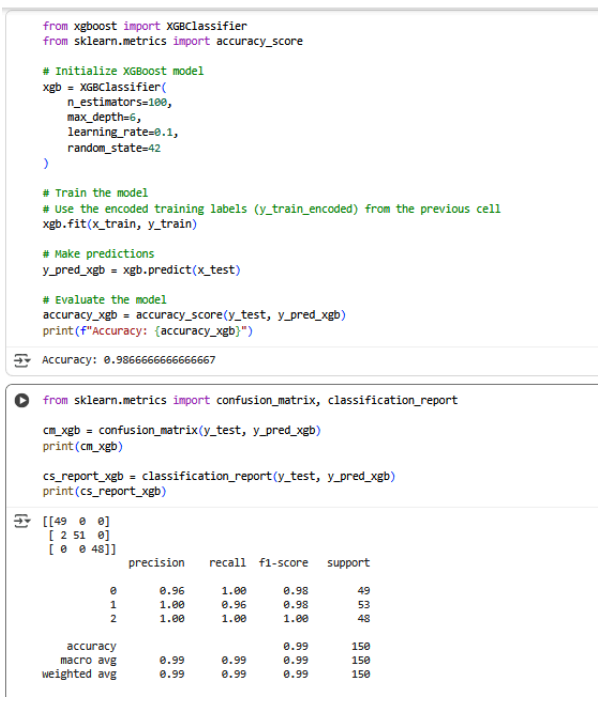
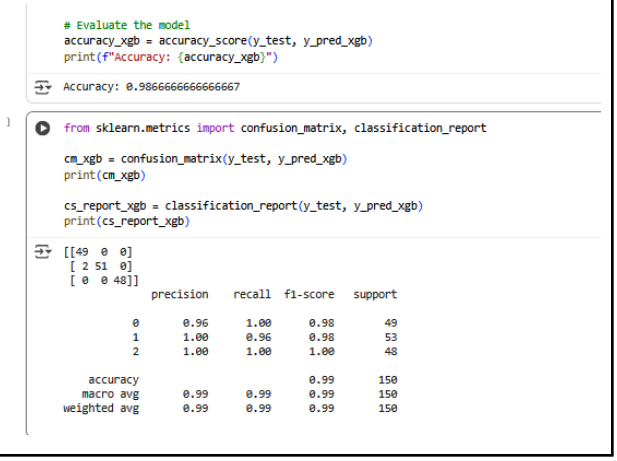
```
y_pred_tabpfn = tabpfn.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred_tabpfn)
accuracy
```

```
1.0
```

```
from sklearn.metrics import confusion_matrix, classification_report

cm_tabpfn = confusion_matrix(y_test, y_pred_tabpfn)
print(cm_tabpfn)

cs_report_tabpfn = classification_report(y_test, y_pred_tabpfn)
print(cs_report_tabpfn)
```

```
[[49  0  0]
 [ 0 53  0]
 [ 0  0 48]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        49
           1       1.00      1.00      1.00        53
           2       1.00      1.00      1.00        48

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

	
Model - 5 : Evaluation	Evaluated models using accuracy, precision, recall, and F1-score. TabPFNClassifier achieved higher performance. 

	`` from sklearn.metrics import accuracy_score accuracy = accuracy_score(y_test, y_pred_tabpfn) accuracy ``` ``` 1.0 ``` ``` from sklearn.metrics import confusion_matrix, classification_report cm_tabpfn = confusion_matrix(y_test, y_pred_tabpfn) print(cm_tabpfn) cs_report_tabpfn = classification_report(y_test, y_pred_tabpfn) print(cs_report_tabpfn) ``` ``` [[49 0 0] [0 53 0] [0 0 48]] precision recall f1-score support 0 1.00 1.00 1.00 49 1 1.00 1.00 1.00 53 2 1.00 1.00 1.00 48 accuracy 1.00 150 macro avg 1.00 1.00 1.00 150 weighted avg 1.00 1.00 1.00 150 ```
Model - 6 : Prediction	Used TabPFNClassifier to predict infant development outcomes for unseen input records. ``` import pandas as pd import numpy as np from tabpfn import TabPFNRegressor from sklearn.model_selection import train_test_split from sklearn.preprocessing import LabelEncoder import pickle # ================================ # 1. Load your dataset # ================================ df = pd.read_csv("infant_development_dataset.csv") # Features (5 inputs matching your Flask form) X = df[["age_months", "height_cm", "weight_kg", "speech_score", "milestone_score"]] # Target (label to predict, e.g., 'period') y = df["period"] # ================================ # 2. Split dataset (train/test) # ================================ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42) # ================================ # 3. Encode target variable (after split) # ================================ # TabPFNRegressor requires numerical targets, so we encode the categorical labels label_encoder = LabelEncoder() y_train_encoded = label_encoder.fit_transform(y_train) y_test_encoded = label_encoder.transform(y_test) # ================================ # 4. Train model # ================================ # Using TabPFNRegressor as requested, but note it's for regression tasks # If you intend classification, use TabPFNClassifier instead. model1 = TabPFNRegressor() model1.fit(X_train, y_train_encoded) # ================================ # 5. Save trained model and label encoder # ================================ # Removed redundant model saving as the user is evaluating the tabpfn model trained earlier. # pickle.dump(model1, open("tabpfn.pkl", "wb")) # pickle.dump(label_encoder, open("label_encoder.pkl", "wb")) # Save the encoder too print(" Train-test split updated to 0.25") print(" y_train and y_test encoded") print(" Model trained with updated split") ``` ``` Train-test split updated to 0.25 y_train and y_test encoded ```

```python
# app.py
from flask import Flask, render_template, request, jsonify, redirect, url_for
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import threading
import time
import os
import requests
from pyngrok import ngrok, conf
import atexit

# Initialize Flask app
app = Flask(__name__)

# Global variables
model = None
label_encoder = None
scaler = None
ngrok_tunnel = None

def create_sample_dataset():
    """Create a sample dataset for the classification problem"""
    print("Creating sample dataset...")

    np.random.seed(42)
    n_samples = 500

    data = []
    for i in range(n_samples):
        period = np.random.choice(['pre_covid', 'during_covid', 'post_covid'],
                                  p=[0.33, 0.34, 0.33])

        if period == 'pre_covid':
            age = np.random.uniform(6, 24)
            height = np.random.uniform(65, 90)
            weight = np.random.uniform(7, 12)
            milestone = np.random.uniform(70, 95)
            speech = np.random.uniform(75, 98)

        elif period == 'during_covid':
            age = np.random.uniform(12, 30)
            height = np.random.uniform(70, 85)
            weight = np.random.uniform(6, 11)
            milestone = np.random.uniform(50, 75)
            speech = np.random.uniform(45, 70)

        else:  # post_covid
            age = np.random.uniform(18, 36)
            height = np.random.uniform(75, 95)
            weight = np.random.uniform(8, 14)
            milestone = np.random.uniform(60, 85)
```

```python
            speech = np.random.uniform(65, 90)

        data.append([age, height, weight, milestone, speech, period])

    df = pd.DataFrame(data, columns=[
        'age_months', 'height_cm', 'weight_kg',
        'milestone_score', 'speech_score', 'period'
    ])

    df.to_csv('infant_data.csv', index=False)
    print("Sample dataset created!")
    return df

def load_model():
    """Load and train the machine learning model"""
    global model, label_encoder, scaler

    try:
        # Load or create dataset
        try:
            df = pd.read_csv('infant_data.csv')
            print("Dataset loaded from infant_data.csv")
        except FileNotFoundError:
            print("Creating new dataset...")
            df = create_sample_dataset()

        # Prepare features and target
        X = df[['age_months', 'height_cm', 'weight_kg', 'milestone_score', 'speech_score']]
        y = df['period']

        # Encode labels
        label_encoder = LabelEncoder()
        y_encoded = label_encoder.fit_transform(y)

        # Scale features
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X_scaled, y_encoded, test_size=0.2, random_state=42
        )

        # Train model
        print("Training Random Forest Classifier...")
        model = TabPFNClassifier(
            n_estimators=100,
            random_state=42,
            max_depth=8
        )
```

```
        model.fit(X_train, y_train)

        # Calculate accuracy
        train_score = model.score(X_train, y_train)
        test_score = model.score(X_test, y_test)

        print(f"Model trained successfully!")
        print(f"   Training accuracy: {train_score:.2%}")
        print(f"   Testing accuracy: {test_score:.2%}")
        print(f"   Classes: {list(label_encoder.classes_)}")

    except Exception as e:
        print(f" Error loading model: {str(e)}")
        raise e

def setup_ngrok():
    """Setup ngrok tunnel"""
    global ngrok_tunnel

    try:
        # Kill existing ngrok processes
        ngrok.kill()

        # Set ngrok authtoken (optional - get from https://dashboard.ngrok.com/get-started/your-authtoken)
        # ngrok.set_auth_token("your_auth_token_here")

        # Create tunnel
        ngrok_tunnel = ngrok.connect(5000)
        public_url = ngrok_tunnel.public_url

        print(" Ngrok tunnel created successfully!")
        print("=" * 60)
        print(f" Your app is now live at: {public_url}")
        print("=" * 60)

        return public_url

    except Exception as e:
        print(f" Error setting up ngrok: {str(e)}")
        print(" Make sure ngrok is installed: pip install pyngrok")
        return None

def cleanup():
    """Cleanup function to close ngrok tunnel on exit"""
    if ngrok_tunnel:
        ngrok.disconnect(ngrok_tunnel.public_url)
        ngrok.kill()
    print(" Ngrok tunnel closed")

# Register cleanup function
atexit.register(cleanup)

@app.route('/')
def home():
```

```
@app.route('/health')
def health_check():
    """Health check endpoint"""
    return jsonify({
        'status': 'healthy',
        'model_loaded': model is not None,
        'classes': label_encoder.classes_.tolist() if label_encoder else None,
        'message': 'Server is running successfully'
    })

@app.route('/predict', methods=['POST'])
def predict():
    """Prediction endpoint"""
    try:
        if model is None:
            return jsonify({
                'status': 'error',
                'message': 'Model not loaded. Please try again in a moment.'
            }), 503

        # Get data from request
        data = request.get_json()

        # Validate input
        required_fields = ['age_months', 'height_cm', 'weight_kg', 'milestone_score', 'speech_score']
        for field in required_fields:
            if field not in data:
                return jsonify({
                    'status': 'error',
                    'message': f'Missing field: {field}'
                }), 400

        # Prepare features
        features = np.array([[
            float(data['age_months']),
            float(data['height_cm']),
            float(data['weight_kg']),
            float(data['milestone_score']),
            float(data['speech_score'])
        ]])

        # Scale features
        features_scaled = scaler.transform(features)

        # Make prediction
        prediction_encoded = model.predict(features_scaled)
        prediction_label = label_encoder.inverse_transform(prediction_encoded)[0]

        # Get probabilities
        probabilities = model.predict_proba(features_scaled)[0]
        confidence = float(np.max(probabilities))

        # Create probability dictionary
        prob_dict = {}
        for i, class_name in enumerate(label_encoder.classes_):
```

```
        # Create probability dictionary
        prob_dict = {}
        for i, class_name in enumerate(label_encoder.classes_):
            prob_dict[class_name] = float(probabilities[i])

        return jsonify({
            'status': 'success',
            'prediction': prediction_label,
            'confidence': confidence,
            'all_probabilities': prob_dict,
            'input_data': data
        })

    except Exception as e:
        return jsonify({
            'status': 'error',
            'message': f'Prediction error: {str(e)}'
        }), 400

@app.route('/model-info')
def model_info():
    """Model information endpoint"""
    if model is None:
        return jsonify({
            'status': 'error',
            'message': 'Model not loaded'
        }), 503

    return jsonify({
        'status': 'success',
        'model_type': 'RandomForestClassifier',
        'classes': label_encoder.classes_.tolist(),
        'features': ['age_months', 'height_cm', 'weight_kg', 'milestone_score', 'speech_score'],
        'description': 'Infant Development Period Classifier'
    })

@app.route('/sample-data')
def sample_data():
    """Provide sample data for testing"""
    samples = [
        {
            'age_months': 18.0,
            'height_cm': 80.0,
            'weight_kg': 10.5,
            'milestone_score': 85.0,
            'speech_score': 90.0,
            'expected_period': 'pre_covid'
        },
        {
            'age_months': 24.0,
            'height_cm': 78.0,
            'weight_kg': 9.5,
            'milestone_score': 65.0,
            'speech_score': 60.0,
            'expected_period': 'during_covid'
        }
```

```
            'milestone_score': 75.0,
            'speech_score': 80.0,
            'expected_period': 'post_covid'
        }
    ]
    return jsonify({'samples': samples})

def start_application():
    """Start the Flask application with ngrok"""
    print(" Starting Flask Application with Ngrok...")

    # Load model first
    print(" Loading machine learning model...")
    load_model()

    # Setup ngrok in a separate thread with delay
    def delayed_ngrok():
        time.sleep(3)  # Wait for Flask to start
        setup_ngrok()

    ngrok_thread = threading.Thread(target=delayed_ngrok)
    ngrok_thread.daemon = True
    ngrok_thread.start()

    # Start Flask app
    print(" Starting Flask server on http://localhost:5000")
    print(" Please wait for ngrok tunnel to be established...")
    app.run(host='0.0.0.0', port=5000, debug=False, threaded=True)

if __name__ == '__main__':
    start_application()
```