Prototype 1:

```
# Smart Irrigation Advisory System (SIAS) - Fixed Version for Google Colab
# Resolves dependency conflicts

# Install required packages with compatible versions
!pip install --upgrade httpx
!pip install requests pandas scikit-learn
!pip install deep-translator  # Alternative to googletrans to avoid conflicts

import pandas as pd
import numpy as np
import requests
import json
from datetime import datetime, timedelta
from sklearn.tree import DecisionTreeClassifier
from sklearn.model selection import train test split
from sklearn.metrics import accuracy_score
import pickle
from deep_translator import GoogleTranslator
import warnings
warnings.filterwarnings('ignore')

class SmartIrrigationAdvisor:
    def   init  (self, weather api key):
        self.weather api key = weather api key
        self.model = None

        # Simple crop water requirements (liters per day per plant)
        self.crop_water_needs = {
            'rice': 5.0,
            'wheat': 3.0,
            'tomato': 2.5,
            'potato': 2.0,
            'onion': 1.5,
            'cotton': 4.0
        }

        # Initialize and train the model
        self._create_training_data()
        self._train_model()

    def _create_training_data(self):
        """Create synthetic training data based on agricultural guidelines"""
        np.random.seed(42)
        n samples = 1000

        # Generate features
        temperature = np.random.normal(28, 8, n samples)  # Celsius
        humidity = np.random.uniform(30, 95, n samples)   # Percentage
        rainfall_forecast = np.random.exponential(2, n_samples)  # mm for next 3 days
        days_since_irrigation = np.random.randint(1, 10, n_samples)
        crop_water_need = np.random.choice(list(self.crop_water_needs.values()), n_samples)

        # Simple rule-based labeling for training
        irrigate = []
        for i in range(n_samples):
            score = 0

            # Hot weather increases need
            if temperature[i] > 32:
                score += 2
            elif temperature[i] > 28:
                score += 1

            # Low humidity increases need
            if humidity[i] < 50:
                score += 2
            elif humidity[i] < 70:
                score += 1

            # Low rainfall forecast increases need
            if rainfall_forecast[i] < 1:
```

```
                score += 2
            elif rainfall forecast[i] < 3:
                score += 1

            # More days since irrigation increases need
            if days_since_irrigation[i] > 5:
                score += 2
            elif days_since_irrigation[i] > 3:
                score += 1

            # High water need crops
            if crop_water_need[i] > 3:
                score += 1

            # Irrigation decision (1 = irrigate, 0 = wait)
            irrigate.append(1 if score >= 4 else 0)

        self.training_data = pd.DataFrame({
            'temperature': temperature,
            'humidity': humidity,
            'rainfall_forecast': rainfall_forecast,
            'days_since_irrigation': days_since_irrigation,
            'crop water need': crop water need,
            'irrigate': irrigate
        })

    def _train_model(self):
        """Train the decision tree model"""
        X = self.training_data.drop('irrigate', axis=1)
        y = self.training_data['irrigate']

        X train, X test, y train, y test = train test split(X, y, test size=0.2, random state=42)

        self.model = DecisionTreeClassifier(max_depth=5, random_state=42)
        self.model.fit(X train, y train)

        # Check accuracy
        y_pred = self.model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print(f"Model trained with accuracy: {accuracy:.2f}")

    def get_weather_data(self, city):
        """Fetch weather data from OpenWeatherMap API"""
        try:
            # Current weather
            current_url =
f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={self.weather_api_key}&units=metric"
            current_response = requests.get(current_url, timeout=10)

            if current_response.status_code != 200:
                print(f"Weather API Error: {current_response.status_code}")
                return self. get dummy weather()

            current_data = current_response.json()

            # Get coordinates for forecast
            lat = current_data['coord']['lat']
            lon = current_data['coord']['lon']

            # 5-day forecast
            forecast url =
f"http://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={self.weather_api_key}&units=metric"
            forecast_response = requests.get(forecast_url, timeout=10)

            # Process current weather
            weather_info = {
                'temperature': current data['main']['temp'],
                'humidity': current_data['main']['humidity'],
                'rainfall_forecast': 0  # Will calculate from forecast
            }

            # Calculate 3-day rainfall forecast
            if forecast_response.status_code == 200:
                forecast_data = forecast_response.json()
```

```
                rainfall = 0
                for item in forecast data['list'][:8]:  # Next 24 hours (3-hour intervals)
                    if 'rain' in item:
                        rainfall += item['rain'].get('3h', 0)
                weather_info['rainfall_forecast'] = rainfall

            return weather_info

        except Exception as e:
            print(f"Error fetching weather data: {e}")
            return self. get dummy weather()

    def _get_dummy_weather(self):
        """Provide dummy weather data for testing when API fails"""
        return {
            'temperature': 30.0,
            'humidity': 60.0,
            'rainfall_forecast': 0.5
        }

    def get_irrigation_advice(self, city, crop, days_since_irrigation):
        """Get irrigation advice for given parameters"""
        # Get weather data
        weather_data = self.get_weather_data(city)

        # Prepare input for model
        crop_water_need = self.crop_water_needs.get(crop.lower(), 2.5)

        input_data = np.array([[
            weather_data['temperature'],
            weather data['humidity'],
            weather data['rainfall forecast'],
            days_since_irrigation,
            crop_water_need
        ]])

        # Get prediction
        prediction = self.model.predict(input_data)[0]
        probability = self.model.predict_proba(input_data)[0]

        # Generate advice
        if prediction == 1:
            confidence = probability[1] * 100
            advice = f"ðŸš° IRRIGATE NOW\nConfidence: {confidence:.0f}%"
            reasoning = f"Current temp: {weather_data['temperature']:.1f}°C, Humidity: {weather_data['humidity']:.0f}%,
Expected rain: {weather_data['rainfall_forecast']:.1f}mm"
        else:
            confidence = probability[0] * 100
            wait_days = min(3, int(probability[0] * 4))
            advice = f"â³ WAIT {wait_days} DAYS\nConfidence: {confidence:.0f}%"
            reasoning = f"Weather conditions are suitable. Rain expected: {weather_data['rainfall_forecast']:.1f}mm"

        return f"{advice}\n\nReasoning: {reasoning}"

    def translate text(self, text, target language='hi'):
        """Translate text to target language using deep-translator"""
        try:
            translator = GoogleTranslator(source='en', target=target_language)
            return translator.translate(text)
        except Exception as e:
            print(f"Translation failed: {e}")
            return text

# Demo version that works without API key for testing
def demo_mode():
    """Demo mode with mock data for testing"""
    print("ðŸŒ¾ SIAS Demo Mode (No API Key Required)")
    print("=" * 50)

    class DemoAdvisor:
        def __init__(self):
            self.crop_water_needs = {
                'rice': 5.0, 'wheat': 3.0, 'tomato': 2.5,
                'potato': 2.0, 'onion': 1.5, 'cotton': 4.0
```

```
            }

        def get_demo_advice(self, crop, days_since_irrigation):
            # Simple demo logic
            crop_water_need = self.crop_water_needs.get(crop.lower(), 2.5)

            # Mock weather conditions
            temperature = 32.0
            humidity = 45.0
            rainfall_forecast = 0.2

            # Simple scoring system
            score = 0
            if temperature > 30: score += 2
            if humidity < 50: score += 2
            if rainfall_forecast < 1: score += 2
            if days_since_irrigation > 4: score += 2
            if crop_water_need > 3: score += 1

            if score >= 4:
                return f"ðŸš° IRRIGATE NOW\nDemo conditions: Hot weather (32°C), Low humidity (45%), Little rain
expected (0.2mm)\nYour {crop} needs water!"
            else:
                return f"â³ WAIT 1-2 DAYS\nDemo conditions: Moderate weather, some rain expected\nYour {crop} can wait
a bit."

    demo_advisor = DemoAdvisor()

    while True:
        print("\n" + "-" * 30)
        crop = input("Enter crop type (rice/wheat/tomato/potato/onion/cotton) or 'quit': ").strip()
        if crop.lower() == 'quit':
            break

        try:
            days = int(input("Days since last irrigation: "))
        except:
            print("Please enter a valid number")
            continue

        advice = demo_advisor.get_demo_advice(crop, days)
        print(f"\nðŸ‘‡ DEMO ADVICE:")
        print("=" * 40)
        print(advice)
        print("=" * 40)

# Main execution
print("ðŸŒ¾ Smart Irrigation Advisory System (SIAS)")
print("=" * 50)

# Check if user wants demo mode or has API key
api key = input("Enter your OpenWeatherMap API key (or press Enter for demo mode): ").strip()

if not api_key:
    demo mode()
else:
    print("\nðŸ”„ Initializing with real weather data...")
    try:
        advisor = SmartIrrigationAdvisor(api_key)

        def run with api():
            print("\nðŸš€ System Ready with Weather API!")
            print("Supported crops: rice, wheat, tomato, potato, onion, cotton")
            print("Type 'quit' to exit\n")

            while True:
                try:
                    print("-" * 30)
                    city = input("Enter your city name: ").strip()
                    if city.lower() == 'quit':
                        break

                    crop = input("Enter crop type: ").strip()
                    if crop.lower() == 'quit':
```

```
                        break

                    try:
                        days_since_irrigation = int(input("Days since last irrigation: "))
                    except:
                        print("Please enter a valid number for days")
                        continue

                    print("\nðŸ”„ Processing...")

                    # Get advice
                    advice = advisor.get_irrigation_advice(city, crop, days_since_irrigation)

                    print(f"\nðŸŒ¾ IRRIGATION ADVICE:")
                    print("=" * 40)
                    print(advice)
                    print("=" * 40)

                except KeyboardInterrupt:
                    print("\nGoodbye! ðŸŒ¾")
                    break
                except Exception as e:
                    print(f"Error: {e}")

        run_with_api()

    except Exception as e:
        print(f"Failed to initialize with API: {e}")
        print("Switching to demo mode...")
        demo_mode()

print("\nâœ… SIAS Demo Complete!")
print("This system helps farmers make irrigation decisions using weather data and ML.")

# Usage Instructions for Google Colab:
"""
ðŸš€ HOW TO USE IN GOOGLE COLAB:

OPTION 1 - Demo Mode (No API needed):
1. Run the code
2. Press Enter when asked for API key
3. Enter crop and days since irrigation
4. Get demo irrigation advice

OPTION 2 - With Weather API:
1. Get free API key from openweathermap.org
2. Run the code
3. Enter your API key when prompted
4. Enter city, crop, and irrigation history
5. Get real-time weather-based advice

FEATURES:
â€¢ Works offline (demo mode)
â€¢ Real weather integration
â€¢ ML-based recommendations
â€¢ Simple interface
â€¢ No dependency conflicts

SUPPORTED CROPS:
rice, wheat, tomato, potato, onion, cotton
"""
```

Prototype 2:

```
# Market Price Advisor (MPA) - Prototype 2
# Fixed version for Google Colab using data.gov.in API

# Install required packages with fixed imports
!pip install pandas requests scikit-learn deep-translator

import pandas as pd
import requests
import numpy as np
from sklearn.linear_model import LinearRegression  # Fixed import
from deep_translator import GoogleTranslator
import datetime
import warnings
warnings.filterwarnings('ignore')

class MarketPriceAdvisor:
    def __init__(self, api_key=None):
        self.api_key = api_key
        self.model = None

        # Commodity mapping for data.gov.in format
        self.commodities = {
            'rice': 'Rice',
            'wheat': 'Wheat',
            'tomato': 'Tomato',
            'potato': 'Potato',
            'onion': 'Onion',
            'cotton': 'Cotton'
        }

        # Demo data for testing without API
        self.demo_data = {
            'rice': [25.5, 26.0, 25.8, 26.2, 26.5, 27.0, 26.8],
            'wheat': [22.0, 22.5, 22.3, 22.8, 23.0, 23.2, 23.1],
            'tomato': [15.0, 16.5, 18.0, 17.5, 19.0, 20.0, 19.5],
            'potato': [12.0, 12.5, 13.0, 12.8, 13.5, 14.0, 13.8],
            'onion': [18.0, 19.0, 18.5, 20.0, 21.0, 19.5, 20.5],
            'cotton': [45.0, 46.0, 45.5, 47.0, 48.0, 47.5, 48.5]
        }

    def fetch_commodity_prices(self, commodity, days=7):
        """Fetch commodity prices from data.gov.in or use demo data"""
        if not self.api_key:
            # Use demo data
            print("Using demo data (no API key provided)")
            prices = self.demo_data.get(commodity.lower(), [20.0] * days)
            dates = pd.date_range(end=datetime.date.today(), periods=days)
            return pd.DataFrame({
                'date': dates,
                'price': prices[-days:],  # Get last 'days' entries
                'commodity': commodity
            })

        try:
            # data.gov.in API endpoint for commodity prices
            # Note: This is a simplified example - actual API structure may vary
            base_url = "https://api.data.gov.in/resource/9ef84268-d588-465a-a308-a864a43d0070"

            params = {
                'api-key': self.api_key,
                'format': 'json',
                'limit': days,
                'filters[commodity]': self.commodities.get(commodity.lower(), commodity)
            }

            response = requests.get(base_url, params=params, timeout=10)

            if response.status_code == 200:
                data = response.json()
                # Process the response based on actual data.gov.in structure
                # This is a simplified processing - adjust based on actual API response
```

```
                records = data.get('records', [])

                if records:
                    df = pd.DataFrame(records)
                    # Assuming the API returns date and price columns
                    # Adjust column names based on actual API response
                    df['date'] = pd.to_datetime(df.get('date', pd.date_range(end=datetime.date.today(),
periods=len(df))))
                    df['price'] = pd.to_numeric(df.get('modal_price', df.get('price', [20.0] * len(df))),
errors='coerce')
                    return df[['date', 'price']].head(days)
                else:
                    print("No data found in API response, using demo data")
                    return self.fetch commodity prices(commodity, days)  # Fallback to demo
            else:
                print(f"API Error: {response.status_code}, using demo data")
                return self.fetch_commodity_prices(commodity, days)  # Fallback to demo

        except Exception as e:
            print(f"Error fetching data: {e}, using demo data")
            return self.fetch_commodity_prices(commodity, days)  # Fallback to demo

    def predict trend(self, df):
        """Use linear regression to predict price trend"""
        if df.empty or len(df) < 3:
            return "Insufficient data for prediction"

        # Prepare data for linear regression
        df = df.copy()
        df['day_num'] = range(len(df))

        # Handle missing values
        df['price'] = df['price'].fillna(df['price'].mean())

        X = df[['day num']].values
        y = df['price'].values

        # Train linear regression model
        self.model = LinearRegression()
        self.model.fit(X, y)

        # Predict next day price
        next day = np.array([[len(df)]])
        predicted price = self.model.predict(next day)[0]

        current_price = df['price'].iloc[-1]
        change_percent = ((predicted_price - current_price) / current_price) * 100

        # Generate advice based on predicted change
        if change_percent > 3:
            advice = f"ðŸ"^ PRICES MAY RISE {change_percent:.1f}% - Consider waiting to sell"
            confidence = "High" if abs(change percent) > 5 else "Medium"
        elif change_percent < -3:
            advice = f"ðŸ"‰ PRICES MAY FALL {abs(change_percent):.1f}% - Sell soon"
            confidence = "High" if abs(change percent) > 5 else "Medium"
        else:
            advice = f"ðŸ"Š PRICES STABLE (Â±{abs(change_percent):.1f}%) - Your choice"
            confidence = "Low"

        return f"{advice}\nConfidence: {confidence}\nCurrent: â,¹{current_price:.2f}, Predicted:
â,¹{predicted price:.2f}"

    def translate_to_hindi(self, text):
        """Translate advice to Hindi"""
        try:
            translator = GoogleTranslator(source='en', target='hi')
            return translator.translate(text)
        except Exception as e:
            print(f"Translation failed: {e}")
            return text

    def get_market_advice(self, commodity, days=7, language='en'):
        """Get complete market advice for a commodity"""
        print(f"\nðŸ"„ Fetching {commodity} prices for last {days} days...")
```

```
        # Fetch price data
        df = self.fetch_commodity_prices(commodity, days)

        if df.empty:
            return "Unable to fetch price data"

        # Show recent prices
        print(f"Recent prices: â‚¹{df['price'].iloc[0]:.2f} to â‚¹{df['price'].iloc[-1]:.2f}")

        # Get trend prediction
        advice = self.predict_trend(df)

        # Translate if needed
        if language == 'hi':
            advice = self.translate_to_hindi(advice)

        return advice

# Demo function for easy testing
def demo_market_advisor():
    """Demo mode for testing without API key"""
    print("ðŸŒ¾ Market Price Advisor (MPA) - Demo Mode")
    print("=" * 50)

    advisor = MarketPriceAdvisor()  # No API key = demo mode

    while True:
        print("\n" + "-" * 30)
        commodity = input("Enter commodity (rice/wheat/tomato/potato/onion/cotton) or 'quit': ").strip()
        if commodity.lower() == 'quit':
            break

        if commodity.lower() not in advisor.commodities:
            print("Supported commodities: rice, wheat, tomato, potato, onion, cotton")
            continue

        try:
            days = int(input("Enter days for trend analysis (3-10): ") or "7")
            if days < 3 or days > 10:
                days = 7
        except:
            days = 7

        language = input("Language (en/hi): ").strip().lower() or 'en'

        advice = advisor.get_market_advice(commodity, days, language)

        print(f"\nðŸ“‹ MARKET ADVICE:")
        print("=" * 40)
        print(advice)
        print("=" * 40)

# Main execution with API key support
def main():
    print("ðŸŒ¾ Market Price Advisor (MPA)")
    print("=" * 50)

    # Get API key from user
    api_key = input("Enter your data.gov.in API key (or press Enter for demo mode): ").strip()

    if not api_key:
        demo_market_advisor()
    else:
        print("\nðŸ”„ Initializing with data.gov.in API...")
        advisor = MarketPriceAdvisor(api_key)

        print("\nðŸš€ System Ready with Real Data!")
        print("Supported commodities: rice, wheat, tomato, potato, onion, cotton")
        print("Type 'quit' to exit\n")

        while True:
            try:
                print("-" * 30)
```

```
                commodity = input("Enter commodity: ").strip()
                if commodity.lower() == 'quit':
                    break

                if commodity.lower() not in advisor.commodities:
                    print("Supported commodities: rice, wheat, tomato, potato, onion, cotton")
                    continue

                try:
                    days = int(input("Days for analysis (3-10): ") or "7")
                    if days < 3 or days > 10:
                        days = 7
                except:
                    days = 7

                language = input("Language (en/hi): ").strip().lower() or 'en'

                advice = advisor.get_market_advice(commodity, days, language)

                print(f"\nðŸ“ˆ MARKET ADVICE:")
                print("=" * 40)
                print(advice)
                print("=" * 40)

            except KeyboardInterrupt:
                print("\nGoodbye! ðŸ‘‹")
                break
            except Exception as e:
                print(f"Error: {e}")

# Run the system
if __name__ == "__main__":
    main()

print("\nâœ… MPA Demo Complete!")
print("This system helps farmers make selling decisions using price trends and ML.")

# Usage Instructions for Google Colab:
"""
ðŸš€ HOW TO USE IN GOOGLE COLAB:

OPTION 1 - Demo Mode (No API needed):
1. Run the code
2. Press Enter when asked for API key
3. Enter commodity and days for analysis
4. Get demo price trend advice

OPTION 2 - With data.gov.in API:
1. Get API key from data.gov.in
2. Run the code
3. Enter your API key when prompted
4. Enter commodity and analysis period
5. Get real-time market advice

FEATURES:
â€¢ Works offline (demo mode)
â€¢ Real commodity price integration
â€¢ ML-based trend prediction (Linear Regression)
â€¢ Simple interface
â€¢ Hindi translation support
â€¢ Uses data.gov.in for Indian market data

SUPPORTED COMMODITIES:
rice, wheat, tomato, potato, onion, cotton

FIXED ISSUES:
â€¢ Corrected sklearn.linear_model import
â€¢ Added data.gov.in API integration
â€¢ Improved error handling
â€¢ Demo mode with realistic price data
"""
```

Prototype 3:

```
# Install required packages
!pip install pandas deep-translator

import pandas as pd
from deep_translator import GoogleTranslator

# Static dataset of schemes (demo data)
SCHEMES = [
    {
        "name": "Pradhan Mantri Fasal Bima Yojana",
        "type": "Subsidy",
        "states": ["All"],
        "crop": ["rice", "wheat", "cotton", "tomato", "potato", "onion"],
        "max_loan_lakh": 2,
        "interest rate": 2.0
    },
    {
        "name": "Kisan Credit Card",
        "type": "Credit",
        "states": ["All"],
        "crop": ["All"],
        "max_loan_lakh": 3,
        "interest rate": 4.0
    },
    {
        "name": "State Agriculture Loan Scheme",
        "type": "Credit",
        "states": ["Maharashtra", "Gujarat", "Punjab"],
        "crop": ["All"],
        "max_loan_lakh": 5,
        "interest_rate": 5.5
    }
]

def lookup schemes(state, crop):
    """Return schemes available for given state and crop."""
    crop = crop.lower()
    matches = []
    for s in SCHEMES:
        if (state in s["states"] or "All" in s["states"]) and (crop in s["crop"] or "All" in s["crop"]):
            matches.append(s)
    return pd.DataFrame(matches)

def check affordability(loan amount, scheme):
    """Simple eligibility: loan_amount <= max_loan_lakh."""
    if loan_amount <= scheme["max_loan_lakh"] * 100000:
        return True
    return False

def translate(text, lang):
    try:
        return GoogleTranslator(source='en', target=lang).translate(text)
    except:
        return text

# Interactive function
def run_credit_advisor():
    print("□ Credit & Subsidy Advisor (CSA)")
    state = input("Enter your state: ").strip()
    crop = input("Enter your crop: ").strip()
    amount = int(input("Enter desired loan amount (in INR): ").strip())
    lang = input("Language (en/hi): ").strip().lower() or 'en'

    df = lookup_schemes(state, crop)
    if df.empty:
        msg = "No schemes found for your state/crop."
        print(translate(msg, lang))
        return

    advice_lines = []
    for _, row in df.iterrows():
```

```
        eligible = check_affordability(amount, row)
        line = f"{row['name']} ({row['type']}), Max Loan: ₹{row['max loan lakh']}L, Interest: {row['interest rate']}% —
"
        line += "Eligible" if eligible else "Not eligible"
        advice_lines.append(line)

    advice = "\n".join(advice_lines)
    print("\n" + translate(advice, lang))

# Run
if __name__ == "__main__":
    run_credit_advisor()
```