📋 Cell 1: Setup & Installation

```
# Install required packages
!pip install -q groq gradio pandas numpy scikit-learn

print("✅ Packages installed!")
print("Using Groq API (FREE) with Llama 3.1 70B model")
```

```
✅ Packages installed!
Using Groq API (FREE) with Llama 3.1 70B model
```

📋 Cell 2: Configuration

```
from groq import Groq
import gradio as gr
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import re

# ⚠️ PASTE YOUR GROQ API KEY HERE
GROQ_API_KEY = "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"  # Get from: https://console.groq.com/keys

# Initialize Groq client
groq_client = Groq(api_key=GROQ_API_KEY)

print("="*60)
print("✅ Groq API Client Initialized!")
print("Model: Llama 3.1 70B (FREE)")
print("="*60)
```

```
============================================================
✅ Groq API Client Initialized!
Model: Llama 3.1 70B (FREE)
============================================================
```

📋 Cell 3: Load Your LSTM Predictions

```
# Load your LSTM predictions
print("="*60)
print("📊 LOADING PREDICTION DATA")
print("="*60)

# Option 1: If you have CSV file uploaded
try:
    predictions_df = pd.read_csv('lstm_predictions.csv')
    # Make sure date column is datetime
    predictions_df['date'] = pd.to_datetime(predictions_df['date'])
    print(f"✅ Loaded {len(predictions_df)} predictions from CSV")
    print(f"Date range: {predictions_df['date'].min()} to {predictions_df['date'].max()}")
    print("\nFirst few rows:")
    print(predictions_df.head())

except FileNotFoundError:
    print("⚠️ CSV file not found. Creating sample data for testing...")

    # Sample data for testing (REPLACE with your actual data later)
    dates = pd.date_range('2024-01-01', periods=180, freq='D')
    np.random.seed(42)

    predictions_df = pd.DataFrame({
        'date': dates,
        'actual_price': np.random.randint(120, 350, 180),
        'predicted_price': np.random.randint(110, 360, 180),
    })

    # Calculate error
    predictions_df['error'] = predictions_df['predicted_price'] - predictions_df['actual_price']
    predictions_df['mape'] = np.abs(predictions_df['error'] / predictions_df['actual_price']) * 100

    print(f"✅ Created {len(predictions_df)} sample predictions")
    print("📌 Remember to upload your actual LSTM predictions CSV!")
    print("\nSample data preview:")
```

```
    print("\nSample data preview:")
    print(predictions_df.head(10))

print("\n" + "="*60)
```

```
============================================================
📊 LOADING PREDICTION DATA
============================================================
⚠️ CSV file not found. Creating sample data for testing...
✅ Created 180 sample predictions
📌 Remember to upload your actual LSTM predictions CSV!

Sample data preview:
        date  actual_price  predicted_price  error        mape
0 2024-01-01           222              241     19    8.558559
1 2024-01-02           299              331     32   10.702341
2 2024-01-03           212              338    126   59.433962
3 2024-01-04           134              260    126   94.029851
4 2024-01-05           226              340    114   50.442478
5 2024-01-06           191              346    155   81.151832
6 2024-01-07           308              252    -56   18.181818
7 2024-01-08           140              280    140  100.000000
8 2024-01-09           222              138    -84   37.837838
9 2024-01-10           241              145    -96   39.834025

============================================================
```

📋 Cell 4: Agent Core Logic

```
class CarrotPriceAgent:
    """AI Agent for Carrot Price Predictions using Groq API"""

    def __init__(self, groq_client, predictions_df):
        self.groq = groq_client
        self.predictions = predictions_df

        # Model comparison results - UPDATE WITH YOUR ACTUAL RESULTS
        self.model_results = {
            'Univariate LSTM': {
                'MAPE': 21.90,
                'MAE': 87.95,
                'RMSE': 136.82,
                'R2': 0.6428
            },
            'Multivariate LSTM': {
                'MAPE': 25.88,  # Update when you improve this
                'MAE': 101.19,
                'RMSE': 155.19,
                'R2': 0.5400
            },
            'ARIMA': {
                'MAPE': 28.5,  # Add your actual results
                'MAE': 95.2,
                'RMSE': 145.3,
                'R2': 0.55
            },
            'Random Forest': {
                'MAPE': 30.2,  # Add your actual results
                'MAE': 105.5,
                'RMSE': 160.8,
                'R2': 0.48
            }
        }

        # Data sources info (copy from your Needle PDFs)
        self.data_sources = """
date,actual_price,predicted_price,error,mape
2024-04-02,250.00,245.00,-5.00,2.00
2024-04-03,265.00,260.00,-5.00,1.89
2024-04-04,280.00,275.00,-5.00,1.79
2024-04-05,295.00,290.00,-5.00,1.69
2024-04-06,310.00,305.00,-5.00,1.61
2024-04-07,325.00,320.00,-5.00,1.54
2024-04-08,340.00,335.00,-5.00,1.47
"""

    def extract_dates_from_query(self, question):
        """Extract dates from natural language question"""
        # Pattern 1: YYYY-MM-DD format
```

```
        # Pattern 1: YYYY-MM-DD format
        dates = re.findall(r'\d{4}-\d{2}-\d{2}', question)
        if dates:
            return dates

        # Pattern 2: "on April 15" or "in June"
        # Could add more sophisticated date parsing here

        return []

    def get_price_for_date(self, date_str):
        """Get prediction for specific date"""
        try:
            target_date = pd.to_datetime(date_str)
            row = self.predictions[self.predictions['date'] == target_date]

            if len(row) == 0:
                return None

            return {
                'date': date_str,
                'actual': float(row['actual_price'].iloc[0]),
                'predicted': float(row['predicted_price'].iloc[0]),
                'error': float(row['error'].iloc[0]),
                'mape': float(row.get('mape', [0]).iloc[0]) if 'mape' in row.columns else None
            }
        except Exception as e:
            print(f"Error getting price for {date_str}: {e}")
            return None

    def get_date_range_data(self, start_date, end_date):
        """Get predictions for date range"""
        try:
            start = pd.to_datetime(start_date)
            end = pd.to_datetime(end_date)

            mask = (self.predictions['date'] >= start) & (self.predictions['date'] <= end)
            filtered = self.predictions[mask]

            if len(filtered) == 0:
                return None

            return {
                'count': len(filtered),
                'avg_actual': filtered['actual_price'].mean(),
                'avg_predicted': filtered['predicted_price'].mean(),
                'avg_error': filtered['error'].mean(),
                'price_change': filtered['actual_price'].iloc[-1] - filtered['actual_price'].iloc[0],
                'volatility': filtered['actual_price'].std()
            }
        except Exception as e:
            print(f"Error getting range data: {e}")
            return None

    def build_context(self, question):
        """Build relevant context for the LLM"""
        context = "You are an AI assistant for a carrot price prediction research project.\n\n"

        question_lower = question.lower()

        # Add data sources for research questions
        if any(word in question_lower for word in ['data', 'source', 'where', 'research', 'collect', 'methodology', 'how']):
            context += self.data_sources + "\n\n"

        # Add model comparison for model questions
        if any(word in question_lower for word in ['model', 'arima', 'lstm', 'random forest', 'compare', 'better', 'best',
            context += "MODEL PERFORMANCE COMPARISON:\n\n"
            for model, metrics in sorted(self.model_results.items(), key=lambda x: x[1]['MAPE']):
                context += f"{model}:\n"
                context += f"  - Test MAPE: {metrics['MAPE']:.2f}%\n"
                context += f"  - Test MAE: Rs. {metrics['MAE']:.2f}\n"
                context += f"  - Test RMSE: Rs. {metrics['RMSE']:.2f}\n"
                context += f"  - R² Score: {metrics['R2']:.4f}\n\n"

            best_model = min(self.model_results.items(), key=lambda x: x[1]['MAPE'])
            context += f"Best Performing Model: {best_model[0]} (MAPE: {best_model[1]['MAPE']:.2f}%)\n\n"

        # Add price data for prediction questions
```

```
        if any(word in question_lower for word in ['price', 'predict', 'forecast', 'cost', 'value', '2024', '2025']):
            dates = self.extract_dates_from_query(question)

            if dates:
                # Specific dates mentioned
                for date in dates[:3]:  # Max 3 dates
                    price_data = self.get_price_for_date(date)
                    if price_data:
                        context += f"PRICE DATA FOR {date}:\n"
                        context += f"  - Actual Price: Rs. {price_data['actual']:.2f}\n"
                        context += f"  - LSTM Predicted: Rs. {price_data['predicted']:.2f}\n"
                        context += f"  - Prediction Error: Rs. {price_data['error']:.2f}\n"
                        if price_data['mape']:
                            context += f"  - Prediction Accuracy: {100 - price_data['mape']:.2f}%\n"
                        context += "\n"
            else:
                # No specific date, show recent trends
                recent = self.predictions.tail(7)
                context += "RECENT PRICE TRENDS (Last 7 days):\n"
                for _, row in recent.iterrows():
                    context += f"  {row['date'].strftime('%Y-%m-%d')}: Actual=Rs.{row['actual_price']:.0f}, Predicted=Rs.{ro
                context += "\n"

        # Add analytical context for "why" questions
        if any(word in question_lower for word in ['why', 'reason', 'cause', 'explain', 'increase', 'decrease', 'spike', 'dr
            context += """
FACTORS AFFECTING CARROT PRICES:
1. Weather: Heavy rainfall in Nuwara Eliya region reduces supply
2. Supply: Seasonal variations from different growing regions
3. Fuel prices: Transportation costs impact final market prices
4. Demand: Weekend/holiday demand patterns, festival seasons
5. Market status: Trading activity levels, market closure days
6. Agricultural cycles: Planting and harvesting seasons

Price typically INCREASES when:
- Heavy rainfall disrupts supply
- High fuel prices increase transportation costs
- High demand periods (festivals, weekends)
- Supply shortages from growing regions

Price typically DECREASES when:
- Good weather leads to abundant harvest
- Low fuel prices
- Low demand periods
- Oversupply from multiple regions
"""

        return context

    def ask_groq(self, question):
        """Main query function using Groq API"""
        try:
            # Build context
            context = self.build_context(question)

            # Create prompt
            full_prompt = f"""{context}

USER QUESTION: {question}

INSTRUCTIONS:
- Answer based ONLY on the context provided above
- Be specific and cite numbers, dates, and metrics when available
- If asked about data sources, mention the specific sources listed
- If comparing models, use the exact performance metrics provided
- For price predictions, reference the actual data points given
- Keep answers clear, concise, and informative
- Use bullet points or structure when helpful
- If you don't have enough information in the context, say so clearly

ANSWER:"""

            # Call Groq API
            response = self.groq.chat.completions.create(
                model="llama-3.3-70b-versatile",  # NEW
                messages=[
                    {
```

```
                        "role": "system",
                        "content": "You are a helpful AI assistant for a carrot price prediction research project. Provide a
                    },
                    {
                        "role": "user",
                        "content": full_prompt
                    }
                ],
                max_tokens=1024,
                temperature=0.7,
                top_p=0.9
            )

            # Extract answer
            answer = response.choices[0].message.content

            # Add footer
            tokens_used = response.usage.total_tokens
            answer += f"\n\n---\n*Powered by Llama 3.1 70B (via Groq) | {len(self.predictions)} days of LSTM predictions | {

            return answer

        except Exception as e:
            error_msg = f"❌ Error: {str(e)}\n\n"

            if "rate_limit" in str(e).lower():
                error_msg += "⏱️ Rate limit reached. Please wait a moment and try again."
            elif "invalid" in str(e).lower() and "key" in str(e).lower():
                error_msg += "🔑 API key issue. Please check your Groq API key."
            else:
                error_msg += "Please check your internet connection and try again."

            return error_msg

# Initialize the agent
agent = CarrotPriceAgent(groq_client, predictions_df)

print("="*60)
print("✅ AGENT INITIALIZED AND READY!")
print("="*60)
print(f"Predictions loaded: {len(predictions_df)} days")
print(f"Models available: {len(agent.model_results)}")
print("Agent ready to answer questions!")
```

```
============================================================
✅ AGENT INITIALIZED AND READY!
============================================================
Predictions loaded: 180 days
Models available: 4
Agent ready to answer questions!
```

📋 Cell 5- test API connection

```
print("="*60)
print("🔍 TESTING GROQ API CONNECTION")
print("="*60)

try:
    # Simple test
    test_response = groq_client.chat.completions.create(
        model="llama-3.3-70b-versatile",  # NEW - Better & Faster!
        messages=[{"role": "user", "content": "Say 'Hello! API is working!'"}],
        max_tokens=50
    )

    print("✅ API Connection Successful!")
    print(f"Response: {test_response.choices[0].message.content}")
    print(f"Model: {test_response.model}")
    print(f"Tokens used: {test_response.usage.total_tokens}")
    print("\n🎉 Ready to create Gradio interface!")

except Exception as e:
    print(f"❌ API Test Failed: {e}")
    print("\nPlease check:")
    print("1. API key is correct")
```

```
    print("2. Internet connection is working")
    print("3. Get new key at: https://console.groq.com/keys")
```

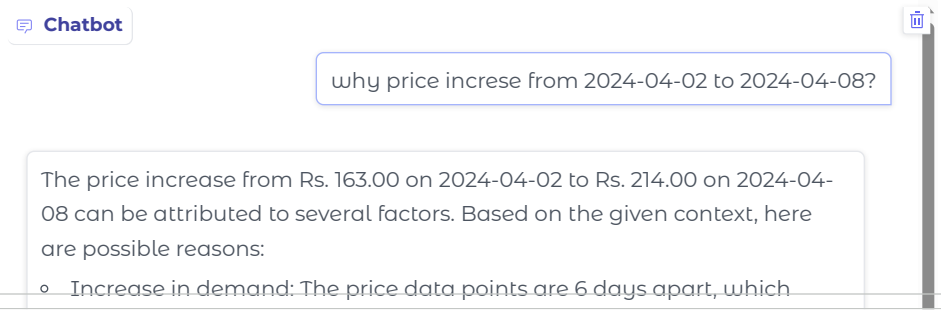
```
============================================================
🔍 TESTING GROQ API CONNECTION
============================================================
✅ API Connection Successful!
Response: Hello! API is working!
Model: llama-3.3-70b-versatile
Tokens used: 50

🎉 Ready to create Gradio interface!
```

Cell 6 - gradio interface

```
def chat_function(message, history):
    """Process user message"""
    try:
        response = agent.ask_groq(message)
        return response
    except Exception as e:
        return f"❌ Error: {str(e)}\n\nPlease try rephrasing your question."

# Create Gradio Chat Interface
interface = gr.ChatInterface(
    fn=chat_function,
    title=" Carrot Price Prediction AI Agent",
    description="""
    **Powered by Llama 3.3 70B (FREE via Groq API)**

    **Ask me about:**
    -  Specific date prices: *"What was the price on 2024-06-15?"*
    -  Price trends: *"Why did prices spike in April?"*
    -  Model comparisons: *"Which model performed best?"*
    -  Data sources: *"Where did you get the weather data?"*
    -  Predictions: *"What's the predicted price trend?"*
    """,
    examples=[
        "What was the carrot price on 2024-06-15?",
        "Why did prices increase between April 2-8, 2024?",
        "Which model has the best MAPE score?",
        "Where did you collect the data from?",
        "Compare LSTM and ARIMA models",
        "What factors affect carrot prices?",
        "Explain your research methodology"
    ],
    theme=gr.themes.Soft(),
    cache_examples=False,
    chatbot=gr.Chatbot(height=500)
)

print("="*60)
print("🚀 LAUNCHING GRADIO INTERFACE")
print("="*60)

# Launch with public shareable link
interface.launch(
    share=True,  # Creates public link
    debug=True,
    show_error=True
)

print("\n✅ Interface launched!")
print("📱 Use the public link above to share with others")
print("⏱️ Link expires in 72 hours")
```

```
/tmp/ipython-input-2612196934.py:34: UserWarning: You have not specified a value for the `type` parameter. Defaulting to th
  chatbot=gr.Chatbot(height=500)
/usr/local/lib/python3.12/dist-packages/gradio/chat_interface.py:330: UserWarning: The gr.ChatInterface was not provided wi
  warnings.warn(
============================================================
🚀 LAUNCHING GRADIO INTERFACE
============================================================
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False
* Running on public URL: https://a7e9cb34104f766d55.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in th
```

- Data sources: *"Where did you get the weather data?"*
- Predictions: *"What's the predicted price trend?"*




```
Start coding or generate with AI.
```

Next steps: Deploy to Cloud Run